Evolutionäre Segmentierung dreidimensionaler Formen unter Verwendung von Satelliten-Seeds

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Kai Engel

Dortmund

2015

Tag der mündlichen Prüfung:	03.02.2015
Dekan:	Prof. Dr.-Ing. Gernot A. Fink
Gutachter:	Prof. Dr. Heinrich Müller
	Prof. Dr. Günter Rudolph

Danksagung

Ich möchte mich ganz herzlich bei Prof. Dr. Heinrich Müller bedanken, der mich als externen Doktoranden aufgenommen und hervorragend betreut hat. Die zahlreichen fachlichen Diskussionen und Besprechungen waren überaus hilfreich. Prof. Dr. Günter Rudolph danke ich herzlich für die Bereitschaft, das Zweitgutachten zu übernehmen. In gleicher Weise gilt mein Dank Prof. Dr. Johannes Fischer für den Vorsitz der Prüfungskommission sowie Dr. Lars Hildebrand für die Mitgliedschaft in der Kommission.

Meinen Freunden und Kollegen danke ich dafür, dass sie stets für mich da waren. Dies beinhaltet auch hilfreiche fachliche Diskussionen sowie die Tatsache, dass sie es verstanden haben, mich immer wieder neu zu motivieren. Explizit hervorheben möchte ich an dieser Stelle Markus für die Einführung in die (und die anfängliche Unterstützung bei der) 3D-Programmierung mit DirectX 9. Dies hat mir sehr geholfen und für eine deutliche Zeitersparnis gesorgt. Des Weiteren danke ich all denjenigen, die für die Evaluation Seed-Punkte auf den Modellen angeordnet haben.

Ein ganz besonderer Dank gilt meiner Freundin Ruth, die mich während der verschiedenen Phasen der Promotion in vielfältiger Weise unterstützt hat. Auch bei meinen Eltern möchte ich mich sehr herzlich bedanken; sie haben mir die bestmögliche Unterstützung zukommen lassen und mich in vielen Lebensbereichen stark entlastet. Dies war mir eine große Hilfe.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Bereich der dreidimensionalen Computergrafik sind je nach Anwendungsgebiet unterschiedliche Arten der Modellrepräsentation üblich. Diese Dissertation betrachtet Modelle, die als Dreiecksnetze gegeben sind. In den Teilbildern (a) und (b) der Abbildung 1.1 ist ein entsprechendes Dreiecksnetz als Drahtgittermodell bzw. als Oberflächenmodell dargestellt. Unter einer *Segmentierung* versteht man die Zerlegung eines solchen Modells entsprechend vorgegebener geometrischer oder semantischer Segmentierungskriterien in einzelne Teile, die *Segmente* genannt werden. Häufig sind bedeutungsvolle Segmente erwünscht, also solche, denen Menschen eine semantische Bedeutung zuschreiben. Ein Beispiel dafür ist in Teilbild (c) der Abbildung 1.1 zu sehen. Für jedes der beiden Ohren, den Kopf sowie den Rumpf des Hasen ist dort das zugehörige Segment farbig hinterlegt.

Segmentierungsverfahren für als Dreiecksnetze gegebene Modelle kommen in verschiedenen Anwendungsgebieten zum Einsatz. Anhand einiger solcher Anwendungsgebiete wird in Abschnitt 1.1 zunächst auf die Bedeutung der Ermittlung „guter“ Segmente eingegangen. Darüber hinaus werden die Eigenschaften automatischer, semi-automatischer und manueller Ansätze gegenübergestellt und auf diese Weise auch die Verwendung

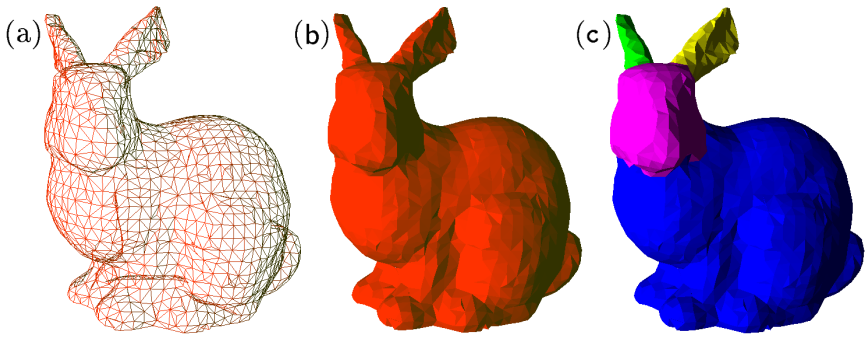


Abbildung 1.1: *Beispiel für ein Dreiecksnetz als Drahtgittermodell (a), als Oberflächenmodell (b) und mit bedeutungsvollen Segmenten (c).*

interaktiver Elemente kurz motiviert. Eine Erläuterung des Gegenstands und der Ergebnisse dieser Dissertation findet in Abschnitt 1.2 statt, bevor in Abschnitt 1.3 auf die verwendete Notation eingegangen wird. In Abschnitt 1.4 werden die Inhalte der Dissertation genannt, die im Rahmen der Forschungstätigkeit veröffentlicht worden sind. Abschließend gibt Abschnitt 1.5 einen Überblick über die Gliederung der nachfolgenden Kapitel.

1.1 Motivation

Bei der *Modellierung* von dreidimensionalen Modellen ist es nicht zwangsläufig notwendig, jedes einzelne Teil des Modells komplett neu zu konstruieren. Vielmehr können prinzipiell auch Teile vorhandener Modelle wiederverwendet werden. So ist es z.B. denkbar, einen Hund zu modellieren, der den Kopf einer Kuh besitzt, indem von einem gegebenen Hundemodell der Rumpf und von einem Kuhmodell der Kopf extrahiert und beide dann geeignet zusammengesetzt werden (vgl. [FKS+04]). Da die Ränder der beiden zu kombinierenden Segmente für gewöhnlich nicht exakt übereinstimmen, werden beim Zusammensetzen zunächst Lücken auftreten, die durch geeignete Ergänzung zusätzlicher Dreiecke zu füllen sind. Darüber hinaus besteht mitunter auch die Notwendigkeit, einzelne Dreiecke eines oder beider Modellteile zu entfernen, um einen guten Übergang zu erzielen. Eine ausführliche Beschreibung dieses Modellierungsvorgangs liefert beispielsweise [FKS+04].

Damit eine solche Konstruktion überhaupt möglich ist, müssen die entsprechenden semantischen Teile[1] verfügbar oder aber leicht aus bestehenden Modellen zu extrahieren sein. Die Segmentierung eines dreidimensionalen Modells ist also der erste wichtige Schritt auf dem Weg zu einem Werkzeug, das eine Konstruktion von neuen Modellen aus einzelnen Bestandteilen bereits vorhandener Modelle ermöglicht. Je besser die Segmentierungsergebnisse sind, umso weniger Arbeit ist sowohl beim manuellen Zusammensetzen als auch bei einer automatischen Konstruktion zu erwarten.

Für die oben genannte Konstruktion ist es hilfreich, die gewünschten Modelle oder Komponenten in einer Modell-Datenbank schnell auffinden zu können. Dies ist unter den Begriffen *3D-Shape Retrieval* oder auch *Content-Based Retrieval* bekannt. Beim Content-Based Retrieval werden anhand eines gegebenen Modells weitere gesucht, die ihm ähnlich sind [ZTS02]. In diesem Zusammenhang spielen semantik-orientierte Segmentierungsverfahren eine wichtige Rolle. So geben beispielsweise Zuckerberger et al. in [ZTS02] ein Verfahren zum Content-Based Retrieval an, bei dem jedes Modell zunächst in wenige semantisch bedeutungsvolle Teile zerlegt wird. Die Formen der einzelnen Teile eines Modells werden dann zur Erzeugung einer geeigneten Signatur verwendet.

Ein weiteres Anwendungsgebiet, bei dem der Segmentierung dreidimensionaler Modelle eine große Bedeutung zukommt, stellt *Texture Mapping* dar. Texture Mapping bezeichnet das Versehen eines dreidimensionalen Modells oder Modellteils mit einer Grafik.

[1] Der Kopf eines Menschenmodells ist ein Beispiel für ein semantisches Teil.

Beispielsweise kann das Modell eines Hasen wie in [LPRM02] oder auch in [AMH02, Bildtafel XLIX] demonstriert mit einer Fell-Textur überzogen werden. Beim Texture Mapping gibt es unterschiedliche Projektionsfunktionen, mit denen eine Textur abgebildet werden kann [AMH02, S. 120]. Welche zu nehmen ist, richtet sich nach dem jeweiligen Modell, das mit einer Textur versehen werden soll. Außerdem existieren oft für die einzelnen Modellteile separate Texturen [LPRM02]. Aus diesen Gründen ist es in der Regel notwendig, ein gegebenes Modell vor der Durchführung des Texture Mapping in seine einzelnen Bestandteile zu zerlegen.

Bei einer *Mesh Simplification* oder *Netzvereinfachung* geht es darum, die Anzahl der Facetten[2] zu reduzieren. Dies kann beim Rendering des Objekts zu einem deutlichen Speedup führen. Häufig reichen reduzierte Oberflächennetze auch vollkommen aus, beispielsweise wenn das zu rendernde Objekt weit vom Betrachter entfernt ist. Eine Netzvereinfachung erfolgt durch sukzessives Entfernen jeweils eines Knotens, gefolgt von einer Neuberechnung der Triangulierung. Bessere Ergebnisse lassen sich erzielen, wenn ein solches Vorgehen jeweils nur auf die einzelnen semantischen Teile des Modells beschränkt wird [ZTS02]. Dadurch bleiben die Übergänge zwischen den Segmenten strukturell erhalten. Somit stellt die Netzvereinfachung ein weiteres Anwendungsgebiet dar, für das Segmentierungsverfahren einen Mehrwert liefern.

Semantik-orientierte Segmentierungsverfahren erweisen sich auch bei der *Kompression von Dreiecksnetzen* (*Mesh Compression*) in einer ganz ähnlichen Weise als nützlich. Allerdings erfolgt die Reduktion der Daten bei der Kompression – anders als bei der Netzvereinfachung – primär durch eine geeignete Kodierung des Netzes. Häufig wird diese zusätzlich mit einer verlustbehafteten Quantisierung kombiniert [Gum00, S. 21]. Wenn die semantischen Teile eines gegebenen Modells jeweils einzeln komprimiert werden, lässt sich mitunter eine bessere Kompressionsrate erzielen, als wenn das gesamte Modell in einem Schritt einer Kompression unterzogen wird. Beispielsweise geben Cheng et al. in [CLJ07] ein entsprechendes Verfahren an, in welchem vor der eigentlichen Durchführung der Kompression die semantischen Teile des Modells ermittelt werden.

Ziel der *Kollisionserkennung* (*Collision Detection*) ist – wie der Name schon andeutet – das Feststellen einer Kollision zwischen zwei oder mehreren Objekten im dreidimensionalen Raum. Dies ist einerseits im Bereich von graphischen Anwendungen wie beispielsweise Computerspielen relevant, auf der anderen Seite aber auch für die Planung sicherer Routen für autonome Roboter. Auch für die Kollisionserkennung liefern Verfahren zur Zerlegung von Dreiecksnetzen in semantisch bedeutungsvolle Segmente einen großen Mehrwert. Beispielsweise nutzen Li et al. in [LWTH01] solche Segmente zur Konstruktion eines Hierarchie-Baumes von Hüllkörpern (Bounding Volumes), der besser für die Kollisionserkennung geeignet ist als Hierarchie-Bäume, die ohne eine solche Zerlegung des Modells in bedeutungsvolle Teile erzeugt worden sind.

Die Ermittlung von Skeletten dreidimensionaler Modelle (*Skeleton Extraction*) ist für einige Bereiche der dreidimensionalen Computergrafik von zentraler Bedeutung. Dies

[2] Als *Facette* wird jedes einzelne Polygon des Dreiecksnetzes, also ein Dreieck, bezeichnet (vgl. Abschnitt 2.2).

gilt insbesondere, da es sich bei dem Skelett um ein Merkmal handelt, welches einen Aspekt der Form des Objekts beschreibt [SPW10]. Auch in diesem Bereich werden verstärkt Segmentierungsverfahren mit herangezogen, die eine semantische Zerlegung des Modells bewirken. Beispielsweise greifen Sun et al. in [SPW10] auf ein Segmentierungsverfahren von Katz et al. [KLT05] zurück. Für jedes ermittelte Segment bestimmen sie anhand eines Merkmalspunktes (Feature Point) und kürzester Wegen von diesem aus zu mehreren Punkten auf den Segmenträndern so genannte *Node-Ringe*. Eine geeignete Interpolation der Mittelpunkte der Node-Ringe wird dann als Skelett genommen.

Unter der *Metamorphose* von Modellen versteht man die Transformation eines dreidimensionalen Modells in ein anderes [LV98]. Segmentierungsverfahren können den Vorgang der Metamorphose erleichtern. Dazu sind Ausgangsmodell und Zielmodell konsistent zu segmentieren. Liegen solche konsistenten Segmentierungen vor, wird jedes einzelne Segment des Ausgangsmodell direkt in das entsprechende Segment des Zielmodells überführt [ZTS02]. Die Betrachtung der Segmente erweist sich als vorteilhaft gegenüber einer Transformation des gesamten Netzes ohne diese Informationen.

Erste Klassifikation von Segmentierungsverfahren: In all den genannten Bereichen kommt der Segmentierung dreidimensionaler Modelle eine große Bedeutung zu. Entsprechend sind in den vergangenen Jahren bereits ganz unterschiedliche Segmentierungsverfahren für Dreiecksnetze entwickelt worden. Die ihnen zugrunde liegenden Ansätze lassen sich in folgende drei Kategorien einteilen:

1. *Vollständig manuelle Segmentierung:* Die Anwendenden selektieren den Bereich des Netzes, der zu einer Komponente gehören soll. Dabei können sie alle Facetten des Dreiecksnetzes markieren, die sie als zu der Komponente zugehörig ansehen, oder sie markieren den Rand der Komponente. Auf jeden Fall handelt es sich um eine vergleichsweise langwierige und mitunter ermüdende Tätigkeit, bei der durch diese Umstände auch nicht immer gewährleistet ist, dass hinreichend akkurate Segmentgrenzen entstehen.

2. *Vollständig automatische Segmentierung:* Das Ausgangsmodell wird automatisch in Segmente zerlegt. Dabei kann allerdings keineswegs sichergestellt werden, dass die ermittelten Segmente den Erwartungen der Anwendenden entsprechen, da diese keinerlei Einfluss auf das Ergebnis nehmen können. Des Weiteren ist je nach Verfahren nicht unbedingt auszuschließen, dass eine Übersegmentierung entsteht. Um „sinnvolle“ Segmente zu erhalten, müssen in einem solchen Fall anschließend – meist dann doch manuell – mehrere der automatisch ermittelten Segmente zu einem einzigen zusammengefasst werden.

3. *Semi-automatische Segmentierung:* Die Ermittlung der gewünschten Segmente läuft im Wesentlichen vollautomatisch ab. Allerdings haben die Anwendenden unterschiedliche Möglichkeiten, das Ergebnis interaktiv zu beeinflussen. Man spricht auch von einer *interaktiven Segmentierung*.

Vollständig automatische Segmentierungsverfahren erfordern keine Interaktion der Anwendenden, liefern aber auch oft keine optimalen Segmente. Mit einer vollständig manuellen Segmentierung lässt sich hingegen zumindest prinzipiell eine beliebig gute Qualität erzielen. Dafür ist ein solches Vorgehen aber relativ zeitaufwändig und bietet somit keinen guten Kompromiss zwischen Aufwand und Ergebnis. Semi-automatische Segmentierungsverfahren vereinigen die Vorteile der beiden anderen. So ist einerseits der Aufwand für die Interaktionen überschaubar und andererseits die Qualität der Segmentierung häufig durchaus zufriedenstellend.

1.2 Gegenstand und Ergebnisse

Die Dissertation liefert neue Beiträge zu zwei Problemkomplexen:

1. Semiautomatische Segmentierung, die überwiegend automatisch arbeitet und effektive Eingriffsmöglichkeiten besitzt.
2. Quantitative Evaluierung des Nutzens manueller Eingriffe.

Die beiden folgenden Unterabschnitte geben eine Übersicht.

1.2.1 Das EvOpSeg-Verfahren

Das in dieser Dissertation präsentierte EvOpSeg-Verfahren – EvOpSeg steht für ***Ev****olutionary* ***Op****timized Mesh* ***Seg****mentation* – arbeitet überwiegend automatisch, erlaubt aber auch manuelle Eingriffe, falls sich dies als notwendig erweist. Ein besonderes Augenmerk beim EvOpSeg-Verfahren liegt darauf, die Anzahl der beinhalteten Interaktionen gering zu halten.

Das EvOpSeg-Verfahren greift das existierende Konzept der Seed-Punkt-basierten Segmentierung auf. Es besteht darin, ausgehend von so genannten Seed-Punkten, die automatisch oder manuell auf der Oberfläche des Modells platziert werden, eine Zerlegung der Oberfläche in jeweils zusammenhängende Stücke durchzuführen, wobei jedes dieser Stücke üblicherweise genau einen Seed-Punkt enthält. Dies kann beispielsweise dadurch erfolgen, dass jede Facette des Dreiecksnetzes auf Basis einer Abstandsfunktion, die implizit ein adäquates Segmentierungskriterium beinhaltet, dem der Facette am nächsten gelegenen Seed-Punkt zugeordnet wird. Die sich so ergebende Zerlegung stellt eine Segmentierung dar (*Seed-Punkt-basierte Segmentierung*). Exemplarisch ist dies in Abbildung 1.2 veranschaulicht. Jede der vier gelben Markierungen im linken Teilbild steht für einen Seed-Punkt. Eine darauf basierende Zerlegung der Oberfläche ist im rechten Teilbild durch unterschiedliche Farben dargestellt.

Für die interaktive Segmentierung ist die Verwendung von Seed-Punkten zur Definition von Segmenten ein günstiger Ansatz, da der Eingabeaufwand gering ist. Ein manuelles

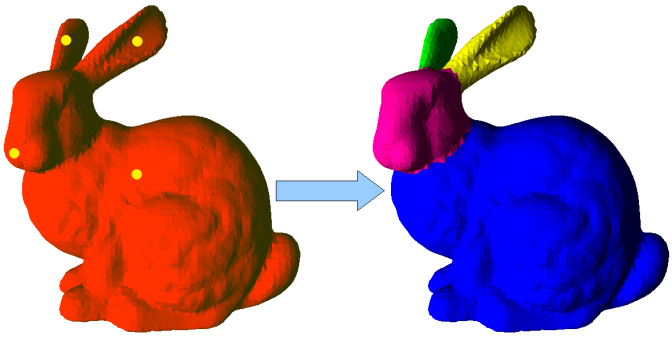


Abbildung 1.2: *Bei einer Seed-Punkt-basierten Segmentierung erfolgt ausgehend von Seed-Punkten (links) eine Zerlegung der Modelloberfläche (rechts).*

Setzen von Seed-Punkten ist für die Anwendenden deutlich einfacher zu bewerkstelligen als das Einzeichnen von Bereichen, innerhalb derer die Segmentgrenzen verlaufen [LHMR08]. Darüber hinaus ist es vergleichsweise leicht, durch Verschiebung der Seed-Punkte gezielt Einfluss auf das Segmentierungsergebnis zu nehmen. Zusätzliche Freiheitsgrade lassen sich realisieren, indem jedem Seed ein Gewicht zugeordnet wird, welches dessen Einfluss angibt [SS08, SNKS09]. Die Verhältnisse der Seed-Gewichte korrelieren tendenziell zu den Größen der entsprechenden Segmente, so dass die Segmentgrenzen auch durch Variation dieser Werte mehr oder weniger gezielt angepasst werden können.

Die Schwierigkeit bei der automatischen Ermittlung von Segmentierungen dreidimensionaler Modelle besteht im Wesentlichen darin, dass lediglich „Low-Level-Informationen“ in Form von Dreiecken und Winkeln zwischen diesen vorhanden sind, aus diesen Informationen aber „High-Level-Strukturen“, nämlich die semantisch bedeutungsvollen Teile, hergeleitet werden müssen [WPP+07]. Anders als bei etablierten Segmentierungsansätzen wird beim EvOpSeg-Verfahren eine Evolutionsstrategie verwendet, um auf diese Weise ohne das Vorhandensein zusätzlicher Informationen semantisch bedeutungsvolle Teile zu erhalten. Nachdem mit möglichst geringem manuellem Aufwand basierend auf Seed-Punkten eine erste grobe Zerlegung des Modells in *Patches* – dies sind Teilstücke des Oberflächennetzes, denen im Gegensatz zu *Segmenten* aber keine semantische Bedeutung zukommen muss – erzeugt worden ist, werden bei Bedarf die Grenzen zwischen den einzelnen Patches automatisch optimiert. Im Wesentlichen besteht das EvOpSeg-Verfahren aus folgenden drei Schritten:

1. *Seed-Platzierung:* Automatisch oder manuell werden Seeds auf dem Modell angeordnet.
2. *Patch-Ermittlung:* Ausgehend von diesen Seeds wird eine Zerlegung des Modells in *Patches* ermittelt.
3. *Patch-Optimierung:* Da berechnete Patches häufig nicht mit semantisch bedeutungsvollen Segmenten übereinstimmen, findet eine evolutionäre Patch-Optimierung statt.

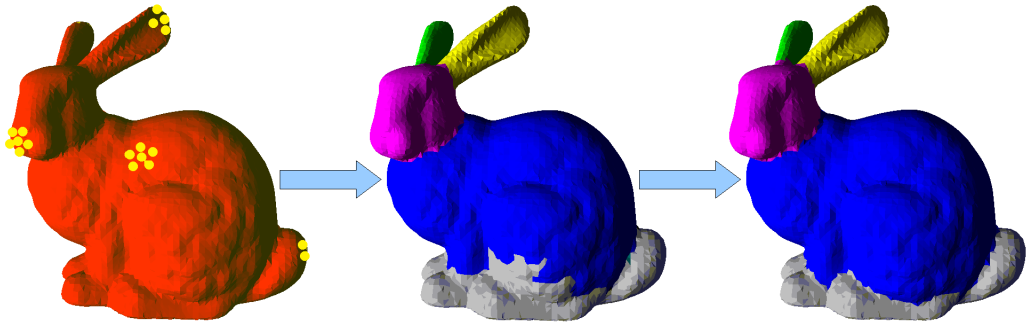


Abbildung 1.3: *Ausgehend von Seed-Punkten (links) wird eine Segmentierung durchgeführt. Die entstandenen Patches (Mitte) werden anschließend optimiert (rechts).*

Diese drei Schritte sind in Abbildung 1.3 veranschaulicht. Sie beinhalten neuartige Ansätze, die nachfolgend zusammengefasst werden.

Seed-Platzierung: Beim EvOpSeg-Verfahren werden gewichtete Seed-Punkte automatisch oder manuell auf dem Modell angeordnet. Es hat sich jedoch gezeigt, dass die übliche Verwendung von Seed-Punkten[3] selbst bei gezielten manuellen Eingriffen in Form von Verschieben der Seeds oder Ändern der Seed-Gewichte oft nicht zu einer optimalen Segmentierung führt. Dieses Problem wird im EvOpSeg-Verfahren durch den neuartigen Ansatz so genannter Satelliten-Seeds angegangen. *Satelliten-Seeds* ermöglichen eine deutlich feinere Einflussnahme, ohne jedoch die Anzahl der Freiheitsgrade zu sehr zu erhöhen. Wie in Abbildung 1.4 dargestellt, erweitern Satelliten-Seeds gewöhnliche Seeds zu speziellen Seed-Clustern, indem die Satelliten-Seeds um die gewöhnlichen Seeds herum angeordnet werden. Solche zusammengehörenden Seeds definieren bei der Patch-Ermittlung gemeinsam ein Segment. Satelliten-Seeds werden sich nicht nur für manuelle Interaktionen, sondern auch für die evolutionäre Patch-Optimierung als äußerst vorteilhaft erweisen.

Patch-Ermittlung: Bei der Patch-Ermittlung hat die Wahl der Abstandsfunktion einen grundlegenden Einfluss auf das Segmentierungsergebnis. Das einfachste Distanzmaß, das einen Abstand auf der Oberfläche angibt, ist die geodätische Distanz, die jedoch Winkel und Krümmungen der Oberfläche nicht direkt berücksichtigt. Eine Erweiterung der geodätischen Distanz um solche winkelbasierten Distanzinformationen ist beispielsweise in [KT03] und [LZSCO09] zu finden. Für das in dieser Dissertation vorgestellte Segmentierungsverfahren sind zwei Distanzfunktionen definiert worden, die sich gut für die Segmentierung eignen:

- *Winkelbasierte Distanz*: In die für das EvOpSeg-Verfahren konstruierte winkelbasierte Distanz fließen neben der geodätischen Distanz auch die Winkel zwischen

[3] Üblicherweise wird pro Segment nur ein einziger Seed-Punkt oder aber eine sehr kleine Menge von Seed-Punkten, die keinen direkten Bezug zueinander haben, verwendet.

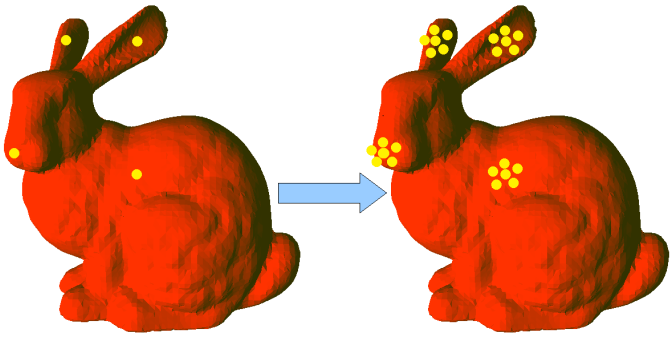


Abbildung 1.4: *Beim Satelliten-Seeding werden gewöhnliche Seeds (links) geeignet um Satelliten-Seeds ergänzt (rechts).*

benachbarten Dreiecken ein. Im Gegensatz zu einer ähnlichen Distanz aus [KT03] werden jedoch ausschließlich konkave Winkel betrachtet und konvexe Winkel vollkommen unberücksichtigt gelassen. Auf diese Weise sollen konkave Objektregionen zu einer größeren Distanz und damit dazu führen, dass die Patch-Grenzen möglichst innerhalb solcher Regionen liegen.

- *Merkmalbasierte Distanz*: Die neu eingeführte merkmalbasierte Distanz erweitert die winkelbasierte Distanz um einen Distanzanteil, der sich aus lokalen Merkmalsinformationen ergibt. Ihr liegt der Gedanke zugrunde, die Länge eines Weges auf der Oberfläche zu vergrößern, wenn dieser über die Grenze zwischen zwei unterschiedlich geformten Bereichen (z.B. einem kugelförmigen und einem ebenen) verläuft. Ein interessanter Aspekt der merkmalbasierten Distanz ist, dass sie auch die Form der einzelnen Modellteile bei der Abstandsberechnung mit zu berücksichtigen versucht, um mit diesen Zusatzinformationen noch bessere Segmentgrenzen zu erzielen, ohne dass die semantischen Teile bereits bekannt sind.

Patch-Optimierung: Gute Segmentierungen sind nach gängiger Auffassung solche, bei denen die Segmentgrenzen möglichst gut durch konkave Modellregionen verlaufen. Dieser Auffassung liegt die von Hoffmann und Richards formulierte *Minima-Rule* zugrunde [HR84]. Als ein zentraler Beitrag der vorliegenden Dissertation wird das Segmentierungsproblem ausgehend von der Minima-Rule in ein Optimierungsproblem überführt, das die folgenden Besonderheiten aufweist:

- Im Rahmen der Optimierung erfolgen nicht nur Änderungen der Seed-Positionen bzw. -Gewichte, sondern auch Anpassungen spezieller Parameter der zur Patch-Berechnung verwendeten Distanzfunktion. Ohne weitere Kenntnisse über das konkrete Modell oder die Modellkategorie kann so eine initial berechnete Segmentierung mit Hilfe einer Evolutionsstrategie optimiert werden.

- Es werden drei Varianten zur Optimierung von Patches angeboten: die direkte Patch-Optimierung, die vom Autor dieser Dissertation in [EM12] vorgestellt

worden ist, die *zweistufige Patch-Optimierung* sowie die *Patch-Optimierung mit gebundener Mutation*. Für jede der Varianten scheint es konkrete Situationen zu geben, in denen sie den anderen beiden überlegen ist.

1.2.2 Evaluierung semi-automatischer Verfahren

Der gängige Ansatz zur Evaluierung der Leistung interaktiver Verfahren ist das Heranziehen von Versuchspersonen. Dies ist mit verschiedenen Schwierigkeiten verbunden. In dieser Arbeit wird ein Ansatz namens *simuliertes manuelles Seeding* präsentiert, der das Verhalten von Versuchspersonen simuliert. Der Ansatz geht von der Annahme aus, dass der Mensch die Seeds abhängig von seiner Kenntnis der angestrebten Segmentierung wählt. Dementsprechend sind bereits segmentierte Modelle die Grundlage des Ansatzes. Unter Verwendung der Segmente werden nach Regeln, die dem Verhalten des Menschen nachvollzogen sind, Seed-Punkte berechnet.

Eine alternative Sichtweise des simulierten manuellen Seeding ist die eines näherungsweise idealen automatischen Seeding. Ihr liegt die Annahme zugrunde, dass menschliche Anwender ein optimales Ergebnis erzielen. In diesem Sinne kann das simulierte manuelle Seeding als untere Schranke für die bestmöglich erzielbare Leistungsfähigkeit von automatischen Seeding-Verfahren betrachtet werden. Allerdings ist das simulierte manuelle Seeding kein Lösungsverfahren, da seine Grundlage ja bereits eine Lösung ist. Das ist mit dem optimalen Seitenersetzungsverfahren von Belady bei Betriebssystemen vergleichbar, das ebenfalls ein theoretisches Konzept darstellt (vgl. [SGG00, S. 315]).

Die wesentliche Voraussetzung für die Evaluierung des EvOpSeg-Verfahrens durch simuliertes manuelles Seeding ist die Existenz von segmentierten Modellen. Motiviert durch eine automatisierte quantitative Beurteilbarkeit der Leistungsfähigkeit automatischer Segmentierungsverfahren wurde in der Vergangenheit eine umfangreiche Sammlung von segmentierten Modellen geschaffen, der *Princeton-Mesh-Segmentation-Benchmark* [CGF09]. Dieser Benchmark bietet auch die Grundlage einer umfangreichen Evaluierung der Segmentierungsgüte des EvOpSeg-Verfahrens. Dabei werden anhand von zwei Metriken, dem Rand-Index und dem Konsistenzfehler, die vom EvOpSeg-Verfahren erzeugten Ergebnisse mit erwarteten Segmentierungen verglichen (vgl. auch [CGF09]). Die beiden Metriken geben eine Art Fehlerwert an, bei dem ein kleiner Wert auf eine gute Segmentierung hinweist. Die Evaluierung, die für alle 380 Modelle des Princeton-Benchmark durchgeführt wurde, betrifft die Güte der automatischen Variante des EvOpSeg-Verfahrens sowie der interaktiven Variante mittels simuliertem manuellen Seeding und mittels Seeding durch Versuchspersonen. Die wesentlichen Ergebnisse sind wie folgt:

- Das Vorgehen zur automatischen Seed-Platzierung führt bei Modellen mit langen, abstehenden Teilen durchaus zu akzeptablen Ergebnissen. Im oben genannten Benchmark waren hingegen auch Modelle enthalten, bei denen das automatische Seeding zu suboptimalen Patches führt. Dennoch liefert das EvOpSeg-Verfahren

bei automatisch platzierten Seeds Segmentierungen, deren über alle Benchmark-Modelle gemittelter Rand-Index-Wert um mehr als 8% besser ist als der entsprechende Durchschnittswert über die Gesamtheit der neun bekannten Verfahren. Bezogen auf den lokalen Konsistenzfehler beträgt die Verbesserung sogar mehr als 14%.

- Das simulierte manuelle Seeding weist eine Verbesserung gegenüber dem durchschnittlichen Rand-Index-Wert der neun Verfahren von über 42% und gegenüber dem durchschnittlichen lokalen Konsistenzfehler von knapp 35% auf.

- Durch Versuchspersonen wurden manuell Seeds auf jeweils mehreren der 380 Benchmark-Modelle angeordnet. Die Ergebnisse, die sich dadurch ergeben haben, sind bezüglich der beiden genannten Metriken besser als die Segmentierungen aller betrachteten etablierten Segmentierungsverfahren. Mit diesen Ergebnissen stimmen die Fehlerwerte recht genau überein, die sich beim simulierten manuellen Seeding ergeben haben. Dies ist ein Indiz dafür, dass das simulierte manuelle Seeding einer echten manuellen Seed-Anordnung durchaus ähnlich ist. Ein Vergleich der Seed-Positionen beim manuellen und beim simulierten manuellen Seeding spricht ebenfalls dafür.

- Es hat sich zeigt, dass das Konzept des Satelliten-Seeding insbesondere bei der evolutionären Patch-Optimierung hilfreich ist. Explizit wird die Auswertung anhand von fünf Modellen in Abschnitt 8.3 gezeigt. Ohne Satelliten-Seeds waren die Rand-Index-Werte stets deutlich schlechter als bei jeder anderen Anzahl von Satelliten-Seeds. Gemittelt über alle untersuchten Modelle ist dieser Fehlerwert in dem Fall, wenn keine Satelliten-Seeds verwendet werden, knapp dreimal so groß wie bei Verwendung von drei Satelliten-Seeds.

1.3 Notation

Im weiteren Verlauf der Arbeit wird größtenteils bewusst zwischen Objekt und Modell unterschieden. Unter einem *Modell* ist dabei die Repräsentation eines *Objekts* aus der Realwelt zu verstehen. Diese Dissertation beschränkt sich auf Modelle, welche die Oberflächen der entsprechenden Objekte mit Dreiecksnetzen approximieren.

Bei dem hier vorgestellten EvOpSeg-Verfahren erfolgt die Berechnung der Segmente ausgehend von Seed-Punkten. Ein *Seed-Punkt* liegt in diesem Kontext stets auf der Oberfläche des Modells. Das Netzdreieck, das den Seed-Punkt enthält, wird *Seed-Dreieck* genannt. Sofern die Unterscheidung zwischen Seed-Punkt und Seed-Dreieck nicht relevant ist, wird in dieser Arbeit auch vereinfachend von *Seed* gesprochen.

Mit $\mathbb{R}$ wird die Menge der reellen Zahlen bezeichnet, mit $\mathbb{R}^+$ die der positiven reellen Zahlen und mit $\mathbb{N}$ die Menge der natürlichen Zahlen ohne die Null; $\mathbb{N}_0$ steht für die Menge der natürlichen Zahlen inklusive der Null. Für zwei reelle Zahlen a und b gibt $[a, b]$ das geschlossene Intervall mit den Endpunkten a und b an.

In dieser Arbeit werden Punkte im $\mathbb{R}^n$ mit $n \geq 2$ mit fetten Buchstaben gekennzeichnet, Knoten eines Graphen mit normalen Buchstaben. Vektoren sind mit einem Vektorpfeil versehen. Die Koordinaten eines Punktes $\mathbf{p}$ bzw. eines Vektors $\vec{v}$ in einem n-dimensionalen Raum werden mit $p_1, p_2, \ldots, p_n$ bzw. $v_1, v_2, \ldots, v_n$ bezeichnet. Für einen Vektor $\vec{v} \in \mathbb{R}^n$ gibt $\|\vec{v}\|$ die euklidische Norm an, d.h. $\|\vec{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2}$. Der (euklidische) Abstand zweier Punkte $\mathbf{p}$ und $\mathbf{q}$ wird durch $\|\vec{p} - \vec{q}\|$ angegeben, wobei $\vec{p}$ und $\vec{q}$ die Stützvektoren zu den Punkten $\mathbf{p}$ und $\mathbf{q}$ sind. Dieser Abstand wird auch mit $d(\mathbf{p}, \mathbf{q})$ bezeichnet. $\langle \vec{p}, \vec{q} \rangle$ gibt das Skalarprodukt der Vektoren $\vec{p}$ und $\vec{q}$ an. Die Benennung von Mengen und Feldern erfolgt mit Großbuchstaben. Die Mächtigkeit einer Menge M wird als $|M|$ notiert. Polygonzüge werden mit Z und ggf. einem Index bezeichnet. Eine Differenzierung unterschiedlicher Polygonzüge erfolgt über die Indizes.

Obere Schranken für den Aufwand von Algorithmen bzw. Komplexitätsfunktionen werden mit Hilfe der O-Notation angegeben (vgl. auch [AM91, S.25]). Für zwei Funktionen $f, g : \mathbb{N} \to \mathbb{R}^+$ bedeutet demnach $f(n) = O(g(n))$, dass $\frac{f(n)}{g(n)}$ durch eine Konstante beschränkt ist, d.h. es gibt ein $n^* \in \mathbb{N}$ und ein $c > 0$, so dass für alle $n > n^*$ gilt: $f(n) \leq c \cdot g(n)$. Das Argument des Maximums $\underset{x \in X}{\operatorname{argmax}} f(x)$ wird im Folgenden als die Menge aller Werte $\bar{x} \in X$ angesehen, für welche die Funktion f ihr Maximum annimmt.

1.4 Publizierte Inhalte

Zentrale Konzepte und Inhalte dieser Dissertation sind im Rahmen der Konferenz *Parallel Problem Solving from Nature - PPSN XII* vorgestellt und in [EM12] veröffentlicht worden. Insbesondere handelt es sich dabei um die Einführung des Konzepts der Satelliten-Seeds, die Definition einer winkelbasierten sowie einer merkmalbasierten Distanzfunktion zur Patch-Ermittlung, die Überführung des Segmentierungsproblems in ein Optimierungsproblem, seine evolutionäre Lösung sowie das simulierte manuelle Seeding. Alle aufgeführten Inhalte sind vom Autor der vorliegenden Dissertation entwickelt und umgesetzt worden. Der Co-Autor der Publikation hat als Betreuer der Dissertation im Verlauf des Promotionsprojekts begleitend Anregungen zur inhaltlichen Ausrichtung gegeben.

1.5 Gliederung

In Kapitel 2 werden alle für die Segmentierung von Dreiecksnetzen grundlegenden Begriffe definiert. Insbesondere wird neben der Formalisierung der Begriffe „Dreiecksnetz“ und „Segmentierung“ auch auf die geodätische Distanz eingegangen. Eine Diskussion verwandter Arbeiten erfolgt in Kapitel 3. Anschließend gibt Kapitel 4 einen Überlick über die einzelnen Bestandteile des in dieser Arbeit vorgestellten Segmentierungsansatzes. Zu diesem gehört unter anderem die Ermittlung einer ersten, noch nicht zwangsläufig optimalen Segmentierung, die *Initialsegmentierung* genannt und in Kapitel 5 beschrieben wird. Auf die neuartige Variante von Seed-Punkten, die *Satelliten-*

Seeds, wird in Kapitel 6 eingegangen. Die evolutionäre Patch-Optimierung ist Gegenstand von Kapitel 7. Im Rahmen der Bearbeitung ist auch eine ausführliche Evaluation erfolgt. Mit dieser beschäftigt sich Kapitel 8. Abschließend fasst Kapitel 9 die erzielten Ergebnisse zusammen und gibt einen Ausblick auf weitere interessante Fragestellungen.

Kapitel 2

Segmentierung von Dreiecksnetzen

Dreidimensionale Modelle können in ganz unterschiedlichen Varianten vorliegen. *Volumenbasierte Modelle* (*solid-based models*) sind „gefüllte“ Modelle, die Informationen über ihr Inneres enthalten und sich dadurch insbesondere für Anwendungen aus dem Bereich des *Computer Aided Design* (CAD) (vgl. [AMH02, S. 439]) eignen. Sie ermöglichen ein syntaktisches Modellieren in Form der *konstruktiven Körpergeometrie* (*Constructive Solid Geometry*) [AM91, S. 279], bei der beispielsweise zwei Objekte voneinander subtrahiert werden können.

Neben volumenbasierten Modellen gibt es auch *oberflächenbasierte Modelle* (*surface-based models*) [AMH02, S. 439]. Diese Art der Modellrepräsentation ist auch als *Boundary Representation* bekannt [WP99, S. 25]. Häufig wird die Oberfläche eines Objekts durch Polygone bzw. insbesondere durch Dreiecke approximiert. Zur Verbreitung von Modellen, die als Polygonnetze gegeben sind, hat sicherlich die Tatsache beigetragen, dass sie recht leicht zu erzeugen sind, beispielsweise durch Abtastung eines Objekts durch Laser-Scanner. Zudem ist das Rendering oberflächenbasierter Modelle sehr performant. Deswegen werden auch volumenbasierte Modelle üblicherweise vor dem Rendering zunächst in Polygonnetze überführt (vgl. [Wat02, S. 48]).

Diese Dissertation befasst sich mit der Segmentierung von Modellen, die in Form von Dreiecksnetzen gegeben sind. Der folgende Abschnitt geht kurz auf Möglichkeiten zur Erzeugung entsprechender dreidimensionaler Modelle ein. Anschließend werden in Abschnitt 2.2 Dreiecksnetze formal definiert sowie der Begriff des Dualgraphen eines Dreiecksnetzes eingeführt. Im Anschluss daran werden in Abschnitt 2.3 unterschiedliche Maße zur Abstandsbestimmung auf Dreiecksnetzen definiert, die für diese Arbeit relevant sind. Abschnitt 2.4 formalisiert den Begriff der Segmentierung im Kontext von Dreiecksnetzen.

2.1 Erzeugung von 3D-Modellen

Die Erzeugung dreidimensionaler Modelle kann auf unterschiedliche Arten erfolgen. Der klassische Weg ist die Konstruktion mit Hilfe eines Modellierungswerkzeugs wie

beispielsweise dem unter GPL-Lizenz stehenden Werkzeug *Blender*[1] oder dem kommerziellen Programm *Maya* von Autodesk[2]. Mit diesen kann das Modell nicht nur komplett neu konstruiert, sondern bei Bedarf auch der Idee aus [FKS+04] folgend aus einzelnen bereits vorhandenen Teilen zusammengesetzt werden. Für bestimmte Modell-Domänen sind spezialisierte Programme entwickelt worden. *MakeHuman*[3] sei hier als ein Beispiel genannt. Es ermöglicht die Erzeugung von Menschenmodellen durch Variation zahlreicher charakteristischer Parameter wie Größe, Alter oder Proportionen der Figur. Zu weiten Teilen geschieht dies über Schieberegler, so dass die Konstruktion sehr intuitiv vonstatten geht und damit wesentlich einfacher ist als bei klassischen Modellierungswerkzeugen. Letztere sind hingegen nicht auf eine einzige Domäne beschränkt, sondern universell einsetzbar.

Auch mit einem 3D-Scanner kann eine Überführung physisch vorhandener Objekte in 3D-Modelle erfolgen. Das Objekt wird dabei von verschiedenen Seiten gescannt und die einzelnen Scan-Ergebnisse, die üblicherweise zunächst als Punktwolken vorliegen, zu einem Gesamtmodell zusammengesetzt. Scanner, die sich zum Digitalisieren wasserdichter Objekte und nicht etwa von Landschaften eignen, lassen sich in Triangulations-Scanner und Structured-Light-Scanner unterteilen. *Triangulations-Scanner* projizieren meist eine dünne Linie auf das Objekt und ermitteln die Tiefeninformationen anhand des Verlaufs der Linie, die von einer zum Laser versetzt befindlichen Kamera aufgenommen wird. Anhand dieser Tiefeninformationen werden einzelne Oberflächenpunkte des Modells ermittelt. Im Gegensatz zu Triangulations-Scannern wird bei *Structured-Light-Scannern* nicht nur eine einzige Linie, sondern ein spezielles Streifenmuster projiziert, welches den vom Projektor aus sichtbaren Teil des Objekts vollständig überdeckt. Auch hierbei erfolgt die Aufnahme durch eine leicht versetzt befindliche Kamera. Anhand der Struktur des aufgenommenen Musters werden die Tiefeninformationen hergeleitet. So ist es möglich, mit einer einzigen Aufnahme die aktuell sichtbare Seite des Objekts zu digitalisieren, weshalb das Structured-Light-Verfahren als sehr schnell gilt.

Bei einigen Realwelt-Objekten ist eine Digitalisierung auch anhand von digitalen Fotoaufnahmen möglich. Man spricht dabei von *Photogrammetrie*. Benötigt werden mehrere Aufnahmen desselben Objekts aus unterschiedlichen Blickrichtungen. Aus den Bildinformationen werden die Positionen rekonstruiert, an denen sich die Kamera beim Ablichten des Modells befunden hat. Manche Verfahren erledigen dies allein anhand charakteristischer Objektmerkmale, die auf mehreren der Einzelbilder vorhanden sind, andere Werkzeuge wie *iModeller 3d*[4] benötigen explizite Zusatzinformationen, zum Beispiel in Form einer Unterlage mit Markierungen, auf der sich das Objekt befindet. Sind die Kamerapositionen inklusive der Ausrichtungen bekannt, kann aus den Bildinformationen eine dreidimensionale Punktwolke konstruiert werden. Wie schon oben erwähnt, wird auch hier aus der Punktwolke ein Oberflächennetz erzeugt. Während konkave Einkerbungen (zum Beispiel Dellen) für 3D-Scanner üblicherweise kein Problem darstellen, versagen hier jedoch Photogrammetrie-Verfahren. Dies liegt daran,

[1] http://www.blender.org
[2] http://www.autodesk.de
[3] http://www.makehuman.org
[4] http://www.imodeller.com

dass das Objekt auf einem Foto nur eine zweidimensionale Schablone für die Definition der Kontur des Modells aus der jeweiligen Blickrichtung angibt. Da dellenartige Einkerbungen auf keinem der Fotos als Kontur auftauchen, wird das Modell an solchen Stellen fehlerhaft sein. 3D-Scannern stehen hingegen in solchen Fällen mit Daten über den Abstand zur Oberfläche mehr Informationen zur Verfügung.

2.2 Dreiecksnetze

Der Fokus dieser Arbeit liegt auf der Segmentierung dreidimensionaler Modelle, die als Dreiecksnetze gegeben sind. Ein erstes Beispiel für solche Netze wurde bereits in Abbildung 1.1 gezeigt. Unter einem Dreiecksnetz wird ein Polygonnetz verstanden, das ausschließlich Facetten mit genau drei Eckpunkten enthält. Es ist analog zur Definition von Polygonnetzen in [FvDFH95, S. 473] wie folgt definiert:

Definition 2.1 *Ein* Dreiecksnetz *ist eine zusammenhängende Fläche, die durch ein Tripel $M = (V, E, T)$ bestehend aus einer Menge $V \subseteq \mathbb{R}^3$ von Eckpunkten, einer Menge $E \subseteq V \times V$ von Kanten und einer Menge $T \subseteq V \times V \times V$ von dreieckförmigen Facetten gegeben ist, so dass gilt:*

1. *Jeder Eckpunkt gehört zu mindestens zwei Kanten.*
2. *Jede Kante gehört zu mindestens einem Dreieck der Menge T und zu höchstens zwei Dreiecken.*
3. *Jedes Dreieck wird von einem geschlossenen Kantenzug bestehend aus drei Kanten der Menge E begrenzt.*

Im Folgenden werden die Facetten eines Dreiecksnetzes gelegentlich auch vereinfachend *Dreiecke* genannt. Die englische Bezeichnung für ein Dreiecksnetz ist *Triangle Mesh*; im Umfeld der Segmentierung von Dreiecksnetzen ist auch die verkürzte Form *Mesh* gebräuchlich, die nachfolgend ebenfalls verwendet wird. Für das in dieser Arbeit vorgestellte Segmentierungsverfahren sind Informationen über die Nachbarschaften der einzelnen Dreiecke essentiell. In diesem Zusammenhang wird auch von *benachbarten Dreiecken* gesprochen.

Definition 2.2 *Zwei Facetten eines Dreiecksnetzes heißen* benachbart *oder* adjazent*, wenn sie eine gemeinsame Kante haben.*

Die Definition eines Dreiecksnetzes (Definition 2.1) beinhaltet keine weiteren Einschränkungen für die Gestalt der Netze. Dreiecksnetze können somit *wasserdicht* sein oder „Löcher“ enthalten bzw. offen sein. Von einem wasserdichten Modell spricht man, wenn es eine durchgängige Oberfläche ohne Lücken hat [GBP07], d.h. wenn es ausschließlich Facetten enthält, die jeweils genau drei Nachbarn besitzen.

Geeignete Datenstrukturen wie die *Winged-Edge*-Datenstruktur von Baumgart [Bau75] oder die *Half-Edge*-Datenstruktur [Män88, S. 161ff.] ermöglichen effiziente Abfragen auf Dreiecksnetzen, beispielsweise die Ermittlung aller angrenzenden Kanten zu einer gegebenen Kante. Im Dateisystem gespeichert werden Dreiecksnetze oft auf die Weise, dass zunächst die Eckpunkte aufgelistet werden, gefolgt von einer Liste der Facetten, wobei für jede Facette auf die drei zugehörigen Eckpunkte verwiesen wird. Die Reihenfolge der zu einer dreieckigen Facette gehörenden Eckpunkte legt die Orientierung der Facette fest, d.h. diese Reihenfolge bestimmt, welches die Außenseite des Dreiecks ist. Üblicherweise ist sie so gewählt, dass die sich daraus ergebenden orientierten Flächennormalen stets nach außen zeigen. Auch in dieser Arbeit wird davon ausgegangen.

Zwei Ebenen im $\mathbb{R}^3$, die einander schneiden, schließen so genannte *dihedrale Winkel* ein. In Abbildung 2.1 sind diese Winkel als α und β eingezeichnet. Sie können wie gewöhnliche Winkel im $\mathbb{R}^2$ bestimmt werden, indem der Schnitt mit einer weiteren Ebenen betrachtet wird, die orthogonal zur Schnittgeraden der beiden zu untersuchenden Ebenen ist [Cox61, S. 129].

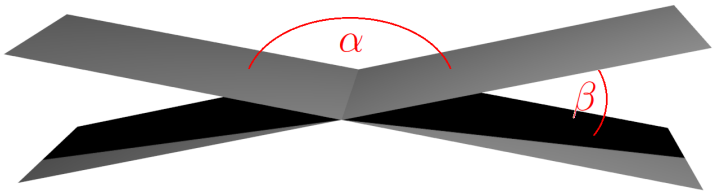


Abbildung 2.1: *Zwei sich schneidende Ebenen führen zu zwei dihedralen Winkeln α und β.*

Auf die gleiche Weise erhält man auch die dihedralen Winkel bei einem Dreiecksnetz. Dort schneiden allerdings im Gegensatz zu Abbildung 2.1 benachbarte Facetten einander nicht, sondern besitzen eine gemeinsame Kante. In Abbildung 2.2 ist in beiden Teilbildern der Schnitt durch zwei solche Dreiecke mit einer zur gemeinsamen Kante senkrechten Ebene dargestellt. An jedem Dreieck (schwarze Linie in der Abbildung) ist dessen Einheitsnormalenvektor $\vec{e}_{n_1}$ bzw. $\vec{e}_{n_2}$ angetragen. Das Innere des Modells ist hellblau angedeutet. Der Winkel β entspricht dem Winkel zwischen den beiden Normalenvektoren. Er lässt sich nach

$$\cos\beta = \langle \vec{e}_{n_1}, \vec{e}_{n_2} \rangle \tag{2.1}$$

berechnen. Daraus lässt sich der im Inneren des Modells liegende Winkel α direkt ableiten, wenn bekannt ist, ob das Modell an der gemeinsamen Kante nach außen (linkes Teilbild von Abbildung 2.2) oder nach innen gewölbt (rechtes Teilbild) ist. Im ersten Fall spricht man von einer *konvexen Kante*, im zweiten von einer *konkaven Kante*.

Die Bestimmung, ob die Kante zwischen zwei benachbarten Dreiecken t_1 und t_2 konvex oder konkav ist, kann wie folgt geschehen: Sei $\vec{e}_{n_1}$ der nach außen zeigende Einheitsnormalenvektor der Facette t_1 und $\vec{g}_1$ der Vektor vom Ursprung zum Schwerpunkt von

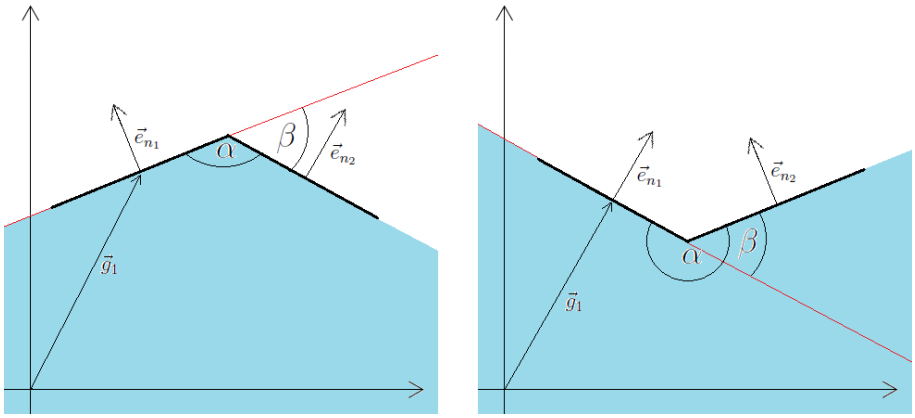


Abbildung 2.2: *Schematische Veranschaulichung des Zusammenhangs zwischen den Lagen der nach außen gerichteten Einheitsnormalenvektoren und dem Vorliegen eines konvexen (linkes Teilbild) oder konkaven (rechtes Teilbild) Winkels zwischen den Facetten.*

t_1 (vgl. Abbildung 2.2). Die Kante zwischen t_1 und t_2 ist *konvex*, wenn der Punkt, der durch die Koordinaten des Vektors $\vec{s} := \vec{g}_1 + \vec{e}_{n_1}$ gegeben ist, nicht auf der gleichen Seite der von t_1 induzierten Ebene liegt wie der Schwerpunkt von t_2. Dies ist im linken Teilbild von Abbildung 2.2 zu erkennen, bei dem die rote Gerade der durch das Dreieck t_1 induzierten Ebene in $\mathbb{R}^3$ entspricht. Um einen *konkaven Winkel* handelt es sich hingegen im rechten Teilbild derselben Abbildung. Dort liegen die beiden angesprochenen Punkte auf derselben Seite der roten Geraden.

Formal lässt sich der Test, ob zwei Punkte auf derselben Seite einer Ebene liegen, mit Hilfe der Normalengleichung der Ebene bestimmen. Diese ist für die durch t_1 in Abbildung 2.2 induzierte Ebene durch

$$\langle \vec{x} - \vec{g}_1, \vec{e}_{n_1} \rangle = 0 \tag{2.2}$$

gegeben. Häufig wird davon ausgegangen, dass die betrachten Normalenvektoren stets so ausgerichtet sind, dass sie vom Nullpunkt wegweisen. Dies ist jedoch bei den in dieser Arbeit betrachteten Dreiecksmodellen nicht zwangsläufig der Fall, da die Einheitsnormalenvektoren – wie oben definiert – vom Inneren des Modells nach außen zeigen und ihre Orientierung nicht etwa von der Lage des Nullpunktes abhängt. Unter dieser Voraussetzung liegt eine konkave Kante vor, wenn $\langle \vec{g}_2 - \vec{g}_1, \vec{e}_{n_1} \rangle > 0$ gilt, wobei $\vec{g}_2$ der Ortsvektor des Schwerpunktes des Dreiecks t_2 ist. Andernfalls ist die Kante konvex. Beim EvOpSeg-Verfahren werden die Werte der Winkel, die an einer konkaven Kante liegen, mit einem positiven Vorzeichen versehen, wohingegen bei konvexen Kanten ein negatives Vorzeichen vorliegt.

Ein Dreiecksnetz kann auch als *Graph* aufgefasst werden, wobei die *Knoten* des Graphen zu den dreidimensionalen Eckpunkten korrespondieren. Die *Kanten* des Graphen ergeben sich dadurch kanonisch in Analogie zu den Kanten des Dreiecksnetzes. Botsch

et al. definieren ein Dreiecksnetz zunächst als Graphen und betten diesen dann in den Raum $\mathbb{R}^3$ ein, indem sie jeden Knoten v_i des Graphen durch

$$\mathbf{p}_i := \mathbf{p}(v_i) = \begin{pmatrix} x(v_i) \\ y(v_i) \\ z(v_i) \end{pmatrix} \in \mathbb{R}^3 \tag{2.3}$$

in einen dreidimensionalen Eckpunkt überführen [BPR+06, S. 18]. Die Unterscheidung zwischen der Netz- und der Graph-Darstellung ist in der vorliegenden Dissertation jedoch nicht primär von Interesse, so dass Eckpunkte eines Dreiecksnetzes gelegentlich auch Knoten genannt werden.

Neben Dreiecksnetzen spielen bei der Segmentierung auch so genannte *Dualgraphen* von Dreiecksnetzen eine Rolle. Sämtliche Knoten des Dualgraphen ergeben sich aus den Netzfacetten: Jedes Dreieck eines gegebenen Dreiecksnetzes M entspricht genau einem Knoten des zugehörigen Dualgraphen G_d. Zu benachbarten Dreiecken korrespondierende Knoten sind in G_d durch eine Kante, die auch *Dualgraph-Kante* genannt wird, verbunden. Dualgraph-Kanten können mit *Kantengewichten* versehen werden. Formal ergibt sich in Anlehnung an [AFS06] folgende Definition:

Definition 2.3 *Gegeben sei ein Dreiecksnetz $M = (V, E, T)$. Ein Graph $G_d = (V_d, E_d)$ mit $E_d \subseteq V_d \times V_d$ heißt* Dualgraph zu M*, wenn es eine eineindeutige Abbildung zwischen der Knotenmenge V_d und der Dreiecksmenge T gibt, so dass zwei Dualgraph-Knoten genau dann durch eine Dualgraph-Kante aus E_d verbunden sind, wenn sie zu einander benachbarten Dreiecken korrespondieren.*

Der Dualgraph eines wasserdichten Dreiecksnetzes beinhaltet nach Konstruktion ausschließlich Knoten mit Grad 3. Ein Beispiel ist in Abbildung 2.3 zu sehen. In der linken Bildhälfte ist der Ausschnitt eines Dreiecksnetzes dargestellt. Die Kanten des zugehörigen Dualgraphen sind im rechten Teilbild schwarz dargestellt und die des ursprünglichen Netzes zusätzlich grau angedeutet.

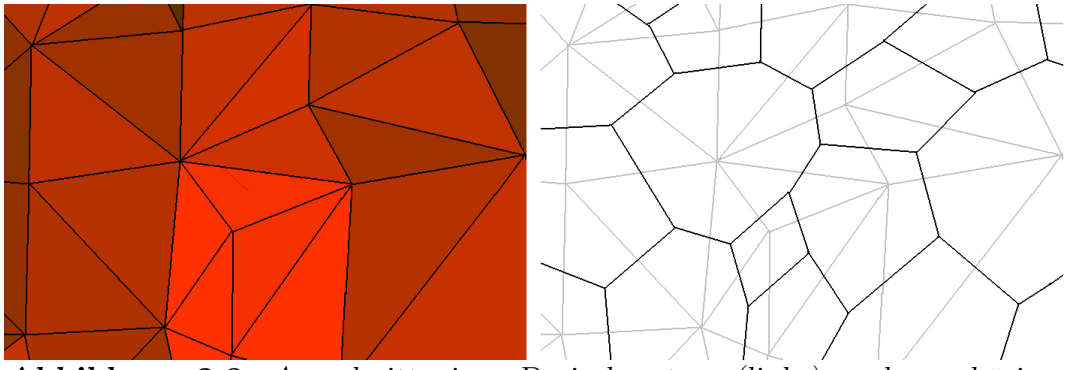


Abbildung 2.3: *Ausschnitt eines Dreiecksnetzes (links) und zugehöriger Dualgraph (rechts).*

Bei Betrachtung eines Dreiecksnetzes als Graph lassen sich Kreise definieren, die für die Anordnung spezieller Seed-Punkte in Kapitel 6 benötigt werden. Ein *Kreis* ist

ein geschlossener Kantenzug, der ausschließlich Kanten des Dreiecksnetzes enthält und keinen Knoten mehr als einmal beinhaltet (vgl. auch [Die06]).

2.3 Distanzen auf Dreiecksnetzen

Klassische Segmentierungsverfahren betrachten häufig Distanzen zwischen Punkten, die auf der Oberfläche des Modells liegen. Da die Oberfläche bei der betrachteten Problemstellung in Form von diskreten Facetten vorliegt, ist insbesondere ein geeignetes Maß für den Abstand der Dreiecke voneinander interessant. Das einfachste solche Maß ist der *geodätische Abstand*[5]. Er ist für zwei auf der Oberfläche liegende Punkte als der kürzeste Pfad zwischen diesen beiden definiert, der ausschließlich auf der Oberfläche verläuft. Das in dieser Dissertation vorgestellte EvOpSeg-Verfahren verwendet eine kombinierte Distanz, in die unter anderem ein geodätischer Anteil einfließt. Da sie über den Dualgraph definiert ist, wird der benötigte geodätische Distanzanteil analog zu der in [Liu09] vorgestellten Konstruktion approximiert, was sich in der folgenden Definition widerspiegelt.

Definition 2.4 *Gegeben seien ein Dreiecksnetz* $M = (V, E, T)$ *sowie zwei darin benachbarte Dreiecke* $t_1, t_2 \in T$. *Sei* $\mathbf{p}$ *der Mittelpunkt der gemeinsamen Kante von* t_1 *und* t_2. *Ferner seien* $\mathbf{p}_{cog}(t_1)$ *der Schwerpunkt von* t_1 *und* $\mathbf{p}_{cog}(t_2)$ *der Schwerpunkt von* t_2. *Dann ist der* geodätische Abstand *der beiden Dreiecke voneinander definiert als*

$$d_{geo}(t_1, t_2) := d(\mathbf{p}_{cog}(t_1), \mathbf{p}) + d(\mathbf{p}, \mathbf{p}_{cog}(t_2)),$$

wobei d *die übliche euklidische Metrik im* $\mathbb{R}^3$ *ist.*

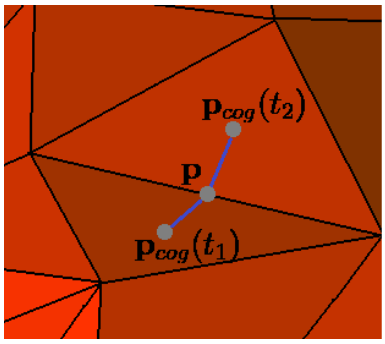


Abbildung 2.4: *Die geodätische Distanz wird für benachbarte Dreiecke mit Hilfe des Mittelpunktes* $\mathbf{p}$ *der gemeinsamen Kante approximiert.*

In Abbildung 2.4 wird die Konstruktion zur Approximation der geodätischen Distanz auf einem gegebenen Dreiecksnetz verdeutlicht. Wie im vorherigen Abschnitt beschrieben, sind die zu den Schwerpunkten benachbarter Dreiecke gehörenden Dualgraph-Knoten durch eine Kante verbunden. Zur Bestimmung der geodätischen Distanz zweier nicht-benachbarter Dreiecke wird jeder solchen Dualgraph-Kante der geodätische

[5] In dieser Dissertation werden die Begriffe *Abstand* und *Distanz* synonym verwendet.

Abstand, den die beiden zu ihren Knoten dualen Punkte auf dem Dreiecksnetz voneinander haben, als Kantengewicht zugewiesen. Mit diesen Gewichten lässt sich die geodätische Distanz beliebiger Dreiecke voneinander approximieren.

Definition 2.5 *Gegeben seien ein Dreiecksnetz $M = (V, E, T)$, der zugehörige Dualgraph $G_d = (V_d, E_d)$, bei dem die Kanten mit den geodätischen Abständen der entsprechenden Dreiecke gewichtet sind, und zwei nicht notwendigerweise benachbarte Dreiecke $t_a, t_b \in T$ mit $t_a \neq t_b$. Die zu den Schwerpunkten von t_a und t_b dualen Knoten seien $v_a, v_b \in V_d$ bezeichnet. Der* geodätische Abstand *der beiden Dreiecke voneinander entspricht der Länge eines kürzesten Weges – bezogen auf die Kantengewichte – im Dualgraphen von v_a nach v_b.*

Neben dem geodätischen Abstand zweier Dreiecke voneinander ist für das EvOpSeg-Verfahren auch ein anderer Abstand relevant, der in dieser Arbeit als *dreieckbasierter Abstand* bezeichnet wird.

Definition 2.6 *Unter dem* dreieckbasierten Abstand *zwischen zwei beliebigen Dreiecken t_a und t_b eines gegebenen Dreiecksnetzes wird die Anzahl der Dualgraph-Kanten verstanden, die im Dualgraph mindestens zu durchlaufen sind, um von v_a nach v_b zu gelangen. Dabei sind v_a und v_b die zu t_a und t_b gehörenden Dualgraph-Knoten.*

Auf dem Dreiecksnetz entspricht der dreieckbasierte Abstand also der Anzahl der Netzkanten, die mindestens zu überqueren sind, um von t_a nach t_b zu gelangen.

2.4 Segmentierung

Unter einer Segmentierung eines Polygonnetzes versteht man die Zerlegung des Netzes in disjunkte Teilnetze, die einander berühren können jedoch nicht „überlappen“. Shamir hat eine Definition für allgemeine Polygonnetze angegeben [Sha07, Sha08]. Übertragen auf Dreiecksnetze ergibt sich folgende Definition:

Definition 2.7 *Sei $M = (V, E, T)$ ein Dreiecksnetz und S eine Menge von Netzelementen mit $S \in \{V, E, T\}$. Eine* Segmentierung Σ *von M ist eine Menge $\Sigma = \{M_0, \ldots, M_{k-1}\}$ von Subnetzen, die von einer Partitionierung von S in k disjunkte Untermengen $S_0, \ldots, S_{k-1}$ induziert wird. Jedes Subnetz M_i, $0 \leq i < k$, stellt ein* Segment *dar.*

Nach dieser Definition resultiert also eine Segmentierung aus einer Partitionierung der Eckpunkte, Kanten oder Dreiecksflächen des gegebenen Netzes. Bei Segmentierungsverfahren, die auf der Partitionierung von Eckpunkten basieren, kann nicht für jede Dreiecksfläche direkt eine Aussage darüber getroffen werden, zu welchem Subnetz sie gehört. Für $k \geq 2$ gibt es nämlich mindestens ein Dreieck, dessen Eckpunkte nicht alle zum gleichen Subnetz gehören. Insgesamt sind drei Fälle zu unterscheiden:

1. Alle Eckpunkte eines Dreiecks gehören zu derselben Partition S_i.
2. Genau zwei Eckpunkte eines Dreiecks gehören zu einer Partition S_i, der dritte Eckpunkt gehört zu einer anderen Partition.
3. Alle drei Eckpunkte eines Dreiecks gehören zu unterschiedlichen Partitionen von S.

Im ersten Fall gehört das Dreieck ebenfalls zum entsprechenden Subnetz. Die anderen beiden Fälle sind nicht so eindeutig. Entweder muss die Dreiecksfläche mit Hilfe von Heuristiken genau einem der Subnetze zugeordnet werden, zu denen die Eckpunkte gehören[6], oder die Dreiecksfläche wird in einem Remeshing-Schritt unterteilt und jede der neuen Teilflächen einem einzigen Subnetz zugeordnet.

Alle drei bei der Partitionierung der Eckpunkte genannten Fälle kommen in Abbildung 2.5 vor. Die Eckpunkte sind in drei Mengen eingeteilt worden, was durch rote, blaue und graue Punkte symbolisiert wird. Die drei Eckpunkte des orange hinterlegten Dreiecks gehören zu drei unterschiedlichen Partitionen. Ohne weiteres ist nicht klar, welcher Partition dieses Dreieck idealerweise zuzuordnen ist. Jede dem orangenen Dreieck benachbarte Facette hat Eckpunkte, die zu genau zwei unterschiedlichen Partitionen gehören; dies entspricht dem zweiten Fall von oben. Darüber hinaus gibt es in diesem Beispiel auch noch einige weitere Dreiecke, die ebenfalls in diese zweite Kategorie fallen.

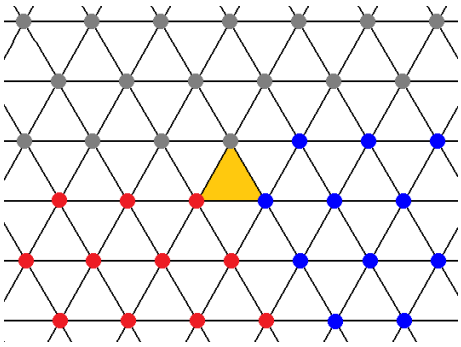


Abbildung 2.5: *Bei einer Partitionierung der Eckpunkte gibt es Dreiecke, die nicht eindeutig einer der Partitionen zugeordnet werden können. Ein solches Beispiel stellt das orange hinterlegte Dreieck dar.*

Ähnliche unklare Situationen in Bezug auf die Zuordnung von Dreiecken zu Segmenten treten auf, wenn eine Partitionierung der Kanten verwendet wird. Es existieren dann Dreiecke, deren drei Kanten nicht zu einer einzigen Partition von S gehören. Anders sieht es hingegen bei einer Partitionierung der Facetten aus. Dies erscheint auch deswegen sinnvoll, da die Dreiecke die einzigen Netzbestandteile sind, die eine echte Oberfläche bilden. Für eine Partitionierung der Dreiecksmenge lässt sich Definition 2.7 dann vereinfachen zu:

[6]Beim zweiten der drei genannten Fälle bietet es sich dabei an, dasjenige Subnetz zu wählen, zu dem zwei der Eckpunkte gehören.

Definition 2.8 *Sei $M = (V, E, T)$ ein Dreiecksnetz. Eine* Segmentierung Σ *von M ist eine Menge $\Sigma = \{M_0, \ldots, M_{k-1}\}$ von Subnetzen, die von einer Partition der Dreiecksmenge T in k disjunkte Untermengen induziert wird. Jedes Subnetz M_i, $0 \leq i < k$, stellt ein* Segment *dar.*

Sofern nicht anders angegeben, wird im Folgenden und insbesondere im Kontext des EvOpSeg-Verfahrens stets diese Definition zugrunde gelegt, wobei zusätzlich gefordert wird, dass jedes einzelne Subnetz zusammenhängend ist. Ein Subnetz M_x gilt dabei als *zusammenhängend*, wenn für zwei beliebige Dreiecke aus M_x stets im Dualgraph des Subnetzes ein Weg zwischen den beiden entsprechenden Dualgraph-Knoten existiert.

In der Literatur werden teilweise unterschiedliche Begriffe für die sich durch die Anwendung eines Segmentierungsverfahrens ergebenden Teile verwendet. Manche Autoren sprechen von *Segmenten* [KLT05, Sha08, GF08, CGF09, SNKS09, Liu09, BAT11], andere von *Komponenten* [LWTH01, KLT05, KJS07, APP+07, Liu09], *Parts* [KJS07, Liu09], *Patches* [AKM+06, AFS06, Sha08, Liu09] oder *Regionen* [LDB03, APP+07]. Teilweise werden einige der Begriffe auch synonym verwendet. Einige Autoren nutzen bewusst mehrere dieser Begriffe, um wie Liu in [Liu09, S. 14] auf unterschiedliche „Segmentierungslevel“ hinzuweisen. Ein Patch wird oft als ein Subnetz angesehen, das topologisch äquivalent zu einer Kreisscheibe ist, d.h. es besitzt nur einen einzigen, zusammenhängenden Rand. In der vorliegenden Arbeit wird allerdings von dieser Sichtweise Abstand genommen. Mit dem Begriff *Patch* wird im Folgenden ein beliebig geartetes, durch das Segmentierungsverfahren berechnetes Teilstück des als Dreiecksnetz gegebenen Modells bezeichnet. Demnach braucht einem Patch keine semantische Bedeutung zu zukommen. Patches, denen hingegen sehrwohl eine solche Bedeutung zukommt oder zumindest zukommen sollte, werden auch als *Segmente* bezeichnet. Unter einer *Komponente* wird ein logisch vorhandener und nicht etwa berechneter Bestandteil des Objekts verstanden.

Sofern nicht der Trivialfall vorliegt, dass ein wasserdichtes Modell in ein einziges Segment unterteilt wird, besitzt jedes Segment mindestens einen Rand. Da die Segmente nicht zwangsläufig topologisch äquivalent zu einer Kreisscheibe zu sein brauchen, kann jedes von ihnen auch mehrere *Segmenträndер* haben. Beispielsweise besitzt ein Segment, das einen Unterarm darstellt, einen Rand zur Hand hin und einen weiteren zur Schulter. Die gemeinsamen Segmenträndер zweier benachbarter Segmente werden hier auch *Segmentgrenzen* genannt. Da sich die Segmentierung nach Definition 2.8 durch eine Partitionierung der Dreiecksmenge ergibt, besteht jeder Segmentrand aus Kanten des Dreiecksnetzes, die im Folgenden *Grenzkanten* genannt werden.

Kapitel 3

Stand der Forschung

In diesem Kapitel wird auf den Stand der Forschung zum Thema „Segmentierung von Dreiecksnetzen“ eingegangen. Der erste Abschnitt gibt zunächst einen Überblick über verwandte Arbeiten. Abschnitt 3.2 geht auf den Ansatz der vorliegenden Arbeit ein und ordnet sie bezüglich der verwandten Ansätze ein. Einige bekannte Segmentierungsverfahren haben sich als „State-of-the-Art-Verfahren“ etabliert. Im Rahmen einer benchmarkbasierten Evaluation wird in Kapitel 8 die Segmentierungsqualität der mit dem EvOpSeg-Verfahren erhaltenen Ergebnisse mit denen dieser etablierten Verfahren verglichen. Eine kurze Vorstellung der relevanten State-of-the-Art-Verfahren erfolgt in Abschnitt 3.3.

3.1 Überblick

Unter der *Segmentierung eines Polygonnetzes* oder *Mesh Segmentation* versteht man eine disjunkte Zerlegung des gegebenen Netzes in mehrere Teilnetze, wobei bestimmte Kriterien, so genannte *Segmentierungskriterien*, zu berücksichtigen sind. Für die einzelnen Teilnetze haben sich, wie in Kapitel 2 erwähnt, unterschiedliche Bezeichnungen etabliert, beispielsweise *Segment* oder *Komponente*. Trotz der unterschiedlichen Bezeichnungen ist in der Literatur jedoch im Wesentlichen das gleiche gemeint. Deswegen wird in diesem Abschnitt im Zusammenhang mit bekannten Verfahren aus Gründen der Verständlichkeit stets der Begriff *Segment* verwendet, solange eine genaue Unterscheidung nicht notwendig ist.

Dreiecksnetze sind spezielle Polygonnetze, denen im Bereich der dreidimensionalen Computergrafik eine besondere Bedeutung zukommt. Beispielsweise eigenen sie sich hervorragend für die Bilderzeugung (Rendering) oder Schnittberechnungen. Nicht zuletzt deswegen befassen sich die meisten der bekannten Mesh-Segmentation-Verfahren mit Dreiecksnetzen. Diese Verfahren lassen sich in zwei unterschiedliche Klassen einteilen [Sha08]: *Surface-Type-Verfahren* und *Part-Type-Verfahren*. Die Verfahren der ersten Klasse werden teilweise auch *Patch-Type-Verfahren* genannt [Sha08]. Bei ihnen erfolgt die Berechnung der Segmente entsprechend bestimmter mathematischer Kriterien (wie

zum Beispiel der Krümmung), die sich üblicherweise auf lokale Netzeigenschaften beziehen. Jedes berechnete Segment ist topologisch äquivalent zu einer Kreisscheibe und wird in der Literatur auch als *Patch* bezeichnet. Dadurch eigenen sich Surface-Type-Verfahren insbesondere für Texture Mapping [SSGH01, Sha08]. Da die Patches jedoch – abgesehen von wenigen Ausnahmen wie etwa bei CAD-Objekten (vgl. [Sha08]) – keine semantischen Teile des Objekts darstellen, sind sie für einige andere Anwendungsdomänen nicht besonders gut geeignet. Abhilfe schaffen *Part-Type-Verfahren*; diese versuchen, das gegebene Objekt (bzw. das dieses Objekt repräsentierende Dreiecksnetz) entsprechend der menschlichen Wahrnehmung in semantische Teile zu zerlegen. Eine semantische Segmentierung eines Pferdemodells wäre beispielsweise durch eine Zerlegung in Kopf, Körper, Beine, Schweif etc. gegeben. Part-Type-Segmentierung ist auch als *semantische Segmentierung* (*semantic segmentation*) [AKM+06, BVLD10] bekannt.

Eine andere mögliche Klassifikation stellt die Unterteilung in *randbasierte* und *regionsbasierte Verfahren* dar [LZHM06]. Bei randbasierten Ansätzen wird in erster Linie der Rand einer Komponente gesucht. Dadurch ist bei Genus-0-Netzen automatisch auch das Innere des entsprechenden Segments definiert. Bei allgemeinen Modellen braucht das hingegen nicht der Fall zu sein. Dort lassen sich durchaus geschlossene Pfade auf der Oberfläche definieren, die keine Grenze zwischen verschiedenen Segmenten darstellen. Ein Beispiel ist in Abbildung 3.1 zu sehen. Im linken Teilbild ist ein Pfad eingezeichnet, der zwei unterschiedliche Objektbereiche voneinander abgrenzt. Auf beiden Seiten des im rechten Teilbild eingezeichneten Pfades, der um das entsprechende Torus-Stück herum führt, befindet sich hingegen dasselbe Segment, d.h. ein solcher Pfad stellt keine Segmentgrenze dar. Anders als bei randbasierten Verfahren werden bei regionsbasierten Segmentierungsverfahren benachbarte Bereiche mit ähnlichen Eigenschaften gesucht, wodurch sich der Segment-Rand automatisch ergibt. Dadurch entstehen stets Ränder, die auch Grenzen darstellen.

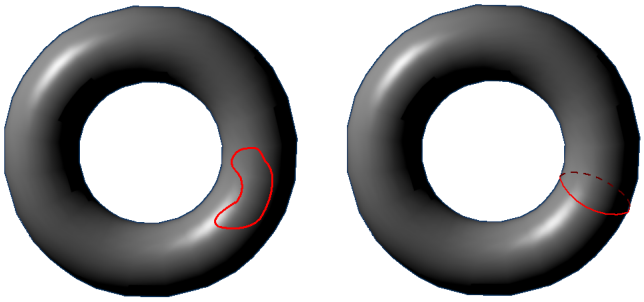


Abbildung 3.1: *Der rot eingezeichnete Pfad im linken Teilbild teilt die Oberfläche des Torus in zwei disjunkte Bereiche. Im rechten Teilbild ist ebenfalls ein geschlossener Pfad eingezeichnet, der die Oberfläche jedoch nicht in zwei Bereiche teilt.*

Einige Segmentierungsverfahren gehen bottom-up vor, andere sind Top-Down-Ansätze (vgl. [WPP+07, Sha08]). *Bottom-Up-Verfahren* starten zunächst mit zahlreichen klei-

nen Patches, die nach und nach zu größeren Teilen zusammengefasst werden. Häufig bestehen die Patches am Anfang jeweils aus lediglich einer einzigen Facette. *Bei Top-Down-Ansätzen* liegt hingegen zu Beginn ein einziges großes Patch vor (das gesamte Polygonnetz), das in zwei oder mehrere Komponenten unterteilt wird. Mit jeder einzelnen dieser Komponenten wird anschließend auf gleiche Weise verfahren. Ein Überblick über entsprechende Verfahren wird beispielsweise in [Sha08] gegeben.

Es kann auch zwischen *automatischen* und *interaktiven Segmentierungsverfahren* unterschieden werden (vgl. Abschnitt 1.1). Weit verbreitet sind vor allem die automatischen Ansätze. Für einen ausführlichen Überblick über diese sei auf [Sha07, Sha08, AKM+06, APP+07] verwiesen. Eine spezielle Form der interaktiven Segmentierung sieht vor, dass die Anwendenden die Grenzen explizit angeben[1]. Da ein solches Vorgehen jedoch zeitintensiv und zudem ausgesprochen ermüdend ist, verwenden interaktive Ansätze häufig Kontrollparameter, die recht einfach manuell zu variieren sind und anhand derer ein Algorithmus die Zerlegung des Modells in Segmente berechnet. Oft wird dazu das Modell mit Strichen, den so genannten *Strokes*, markiert, welche die einzelnen Segmente definieren. Je nach Verfahren können die Strokes, wie beispielsweise in [ZWC+10] beschrieben, innerhalb der jeweiligen Segmente platziert werden, oder entlang der Grenzen zwischen den einzelnen Objekt-Teilen [FKS+04] bzw. quer zu diesen Grenzen [ZT10] – d.h. die Grenzen schneidend – liegen.

Den meisten bisherigen Ansätze – unabhängig davon, ob es sich um automatische oder interaktive handelt – liegen konkrete Heuristiken zugrunde, die als Segmentierungskriterien herangezogen werden; diese sind für alle möglichen Arten von Modellen identisch. Im Gegensatz dazu stellen Kalogerakis et al. [KHS10] einen Ansatz vor, bei dem automatisch – ausgehend von einer Menge markierter Trainingsmodelle – geeignete Parameterwerte erlernt werden. Der Lernprozess ist dabei unabhängig von dem zu segmentierenden Dreiecksnetz. Dieser Ansatz bietet eine flexible Möglichkeit, domänenspezifische Parameterwerte und damit unterschiedliche Arten von Segmentierungen für verschiedene Modellkategorien zu erlernen, ohne eine manuelle Parameteranpassung vornehmen zu müssen. Wenngleich der überwachte Lernvorgang für eine konkrete Modellkategorie lediglich einmal, in einem Vorverarbeitungsschritt, durchzuführen ist, ändert dies nichts an der Tatsache, dass sehr viel Rechenzeit benötigt wird.

3.2 Ansatz dieser Arbeit

Eine Zielsetzung dieser Arbeit ist die Entwicklung und Implementierung eines neuartigen interaktiven Verfahrens zur Zerlegung von dreidimensionalen Modellen in Segmente mit semantischer Bedeutung, das möglichst auch bei solchen Modellkategorien gute Ergebnisse liefert, bei denen bekannte Verfahren Schwächen aufweisen. Zur Erlangung semantischer Segmentierungen ist es notwendig, die menschlichen Wahrnehmungen in Kriterien zu überführen, welche algorithmisch im Programm verankert werden können.

[1] Streng genommen handelt es sich nicht wirklich um ein Segmentierungsverfahren im eigentlichen Sinne, da die Segmentierung ausschließlich von den Anwendenden durchgeführt wird.

Zwei häufig verwendete Kriterien kommen ebenfalls in dieser Arbeit zum Einsatz: die so genannte *Minima-Rule* [HR84] und ein „Glattheitskriterium“. Die Minima-Rule besagt, dass Grenzen zwischen realen Objektteilen typischerweise weitgehend entlang konkaver Kanten bzw. durch konkave Oberflächenregionen verlaufen. Ihr liegt die Beobachtung zugrunde, dass üblicherweise eine konkave Grenzkontur entsteht, wenn zwei dreidimensionale Objekte einander durchdringen. Ein Beispiel ist in Abbildung 3.2 zu sehen: Zwei einander durchdringende Kugeln bilden ein Schneemann-ähnliches Objekt. Ist eine Zerlegung des Objekts in zwei Stücke gesucht, sollte der Schnitt offensichtlich durch den konkaven Bereich verlaufen.

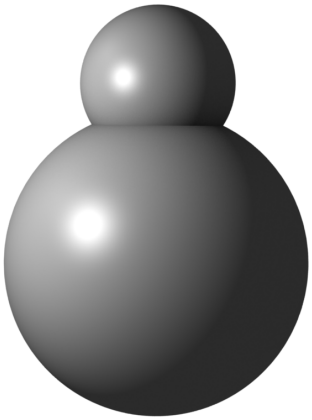


Abbildung 3.2: *Ein aus zwei einander durchdringenden Kugeln konstruiertes dreidimensionales Modell.*

Das oben genannte „Glattheitskriterium“ fordert glatte, d.h. nicht allzu stark „ausgefranste“ Segmentgrenzen. Es ist unwahrscheinlich, dass die beiden angesprochenen sowie gegebenenfalls weitere Kriterien an die Fähigkeiten der Menschen heranreichen, semantisch korrekte Grenzen zwischen einzelnen Objektteilen zu erkennen. Das ist einer der Gründe, weswegen automatische Ansätze häufig keine optimalen Ergebnisse liefern. Es scheint somit sinnvoll zu sein, zusätzlich manuelle Eingriffsmöglichkeiten zu erlauben, wobei der automatische Anteil jedoch so ausgelegt sein sollte, dass möglichst wenige manuelle Eingriffe notwendig sind.

Seed-Punkte stellen eine gute Möglichkeit dar, sowohl Segmente automatisch zu berechnen, als auch interaktive Eingriffsmöglichkeiten zu bieten. Es existieren mehrere Segmentierungsverfahren, die auf einem Seed-Punkt-Prinzip basieren [KJS07, LHMR08, MPS$^+$04, SS08, SNKS09]. Solche Verfahren gehen regionsbasiert vor: Jeder Seed-Punkt definiert eine Netzregion, die ihn umgibt und die als Segment genommen wird; jedes Element einer derartigen Region kann dann als dem entsprechenden Seed-Punkt zugeordnet angesehen werden. Sofern es keine Bereiche des Dreiecksnetzes gibt, die keinem Seed-Punkt zugeordnet sind, liegt eine vollständige Zerlegung des Netzes in Regionen – und damit eine Segmentierung – vor. Es existieren unterschiedliche Möglichkeiten, ausgehend von Seed-Punkten eine Zerlegung in Segmente zu bestimmen. Das in

dieser Arbeit vorgestellte EvOpSeg-Verfahren basiert auf Seed-Punkten und erzeugt Segmente auf ähnliche Weise wie es Simari et al. in [SS08, SNKS09] mit gewichteten Voronoi-Centern gemacht haben. Allerdings nutzen Simari et al. ein Dreiecksnetz, das aus dem ursprünglichen durch *Multidimensionale Skalierung* (*MDS*) [Hor69] entstanden ist, und betrachten räumliche Distanzen. Das EvOpSeg-Verfahren verwendet hingegen Distanzen auf der Oberfläche des Modells. Katz und Tal [KT03] haben eine Kombination aus winkelbasierter Distanz, bei der der dihedrale Winkel zwischen benachbarten Dreiecken berücksichtigt wird, und geodätischer Distanz definiert. Liu et al. [LZSCO09] haben diesen Distanzbegriff um so genannte Volumetric Shape Images erweitert. Beim EvOpSeg-Verfahren spielen neben der geodätischen Distanz zwei weitere Distanzen eine zentrale Rolle: Bei der ersten handelt es sich ähnlich wie in [KT03] um eine Kombination aus geodätischer und einer winkelbasierten Distanz, während bei der zweiten zusätzlich auch Formmerkmale berücksichtigt werden.

Bei einer *bedeutungsvollen Segmentierung* müssen die Segmente bzw. deren Grenzen gewisse Kriterien erfüllen, also zum Beispiel der oben genannten Minima-Rule genügen. Seed-Punkt-basierte Verfahren eignen sich hervorragend für manuelle oder automatische Optimierungen, die angewendet werden können, um eine Lösung zu erhalten, welche die Kriterien möglichst gut erfüllt. Solche Optimierungen lassen sich beispielsweise durch ein geeignetes Verschieben der Seed-Punkte realisieren. Die Optimierung wird dadurch begünstigt, dass jede Region in der Regel durch lediglich einen einzigen Seed-Punkt definiert und somit die Dimension des Parameterraums verhältnismäßig klein ist. Häufig reicht es nämlich aus, nur wenige Seed-Punkte zu verschieben, um den gewünschten Effekt zu erzielen. Andererseits führen wenige Seed-Punkte mitunter dazu, dass einige der berechneten Segmentgrenzen subjektiv als „schlecht“ angesehen werden. Als Abhilfe für diese Problem nutzen manche Verfahren eine Boundary-Smoothing-Technik in einem Nachbearbeitungsschritt. Ein ausführlicher Überblick mit einem Beispiel ist in [KT09] zu finden. Boundary-Smoothing-Techniken arbeiten üblicherweise lokal, d.h. wenn eine berechnete Segmentgrenze weit von der eigentlichen Komponentengrenze entfernt liegt, ist Boundary-Smoothing wenig hilfreich. Eine weitere Möglichkeit ist durch Betrachtung einer Übersegmentierung mit geeignetem Zusammenfassen einzelner Segmente gegeben. Der Nachteil hieran ist jedoch, dass mehr Freiheitsgrade vorliegen, was die Effizienz reduzieren kann.

In der vorliegenden Arbeit wird ein neuartiger Satelliten-Seeding-Ansatz vorgestellt, mit dem die oben genannten Probleme vermieden werden sollen. Dabei werden spezielle *Seed-Cluster* verwendet. Jeder Seed eines Clusters definiert einen Teil des entsprechenden Segmentes oder Patches. Im Kontext des Satelliten-Seeding wird jeder solche Teil in den folgenden Kapiteln als *Sub-Patch* bezeichnet. Die Sub-Patches aller Seeds eines Clusters bilden zusammen das Patch. Obwohl somit beim Satelliten-Seeding eine Art Übersegmentierung genutzt wird, ist in diesem Fall dennoch klar definiert, welche Seed-Punkte zusammengehören. Dies stellt einen bedeutenden Unterschied zu klassischen Übersegmentierungen dar.

3.3 Evaluation von Segmentierungsverfahren

Anfangs wurde die Qualität von Segmentierungsergebnissen lediglich visuell beurteilt, ein objektiver Vergleich der Ergebnisse war auf diese Weise jedoch nicht möglich. Erst im Jahr 2009 haben Benhabiles et al. sowie Chen et al. Benchmarks veröffentlicht, die eine möglichst objektive Bewertung der Segmentierungsqualität erlauben [BVLD09, CGF09]. Diese Benchmarks enthalten neben zahlreichen Modellen auch mehrere manuell erzeugte Segmentierungen eines jeden Modells, die als optimale Segmentierungen angesehen und *Ground-Truth-Daten* oder *Ground-Truth-Segmentierungen* genannt werden. Der Benchmark-Datensatz von Chen et al., der auch *Princeton-Mesh-Segmentation-Benchmark* oder kurz *Princeton-Benchmark* genannt wird, ist wesentlich umfangreicher als der andere oben angesprochene. Er enthält zusätzlich die Segmentierungsergebnisse von sieben bekannten Segmentierungsverfahren, die zu den Standard-Verfahren gehören [STK02, LHMR08, AFS06, GF08, KLT05, SSCO08]. Chen at al. haben Metriken angegeben, mit denen die Abweichung einer berechneten Segmentierung von einer manuell erzeugten quantifiziert werden kann [CGF09]. Sie haben außerdem die Ergebnisse der sieben Standard-Verfahren entsprechend dieser Metriken bewertet. Auch in dieser Dissertation wird eine Bewertung der Ergebnisse mit Hilfe des Princeton-Benchmark durchgeführt. Ergänzend zu [CGF09] ist ein Überblick über häufig verwendete Metriken zur Bewertung von Dreiecksnetz-Segmentierungen [BVLD10] zu entnehmen.

Im Folgenden werden einige Verfahren zur Segmentierung dreidimensionaler Modelle kurz beschrieben, die als etabliert gelten und mit deren Ergebnissen die Resultate des EvOpSeg-Verfahrens im weiteren Verlauf der Arbeit verglichen werden.

„Randomized Cuts“-Segmentierung

Beim Randomized-Cuts-Verfahren [GF08] wird der Dualgraph des Netzes genutzt. Jeder Dualgraph-Kante sind zwei Kostenarten zugeordnet: *Traversal-Cost* und *Cut-Cost*. Niedrige Cut-Costs korrespondieren zu guten Segmentgrenzen. Niedrige Traversal-Cost-Pfade entstehen im Dualgraphen, wenn die entsprechenden Regionen der Mesh-Oberfläche mit großer Wahrscheinlichkeit zu der gleichen Komponente gehören. Beide Kosten werden mit Hilfe eines „Konkavitätsgewichts“ $w(\theta) := \min((\frac{\theta}{\pi})^{\alpha}, 1)$ bestimmt, das für konkave Kanten gegen 0 geht und für konvexe Kanten 1 beträgt und damit deutlich größer ist. Dabei ist θ der außenliegende dihedrale Winkel, der an der jeweiligen Kante vorliegt. Golovinskiy und Funkhouser haben stets $\alpha = 10$ gewählt [GF08]. Die Cut-Costs ergeben sich durch Multiplikation der Länge der Netzkante mit $w(\theta)$. Traversal-Costs ergeben sich hingegen durch Division des Abstands der Mittelpunkte der Facetten, die zu den beiden Endpunkten der Dualgraph-Kante korrespondieren, durch $w(\theta)$.

Das Randomized-Cuts-Verfahren nutzt unterschiedliche Algorithmen zur Erstellung von Segmentierungen, wobei die Wahl zufällig erfolgt. Genauer beschrieben haben die Autoren mit *K-Means Clustering* [STK02], *Hierarchical Clustering* und *Min*

Cut [KT03] drei Segmentierungsverfahren. Der Randomized-Cuts-Ansatz besteht aus vier zentralen Schritten (vgl. [GF08]):

1. Im *ersten Schritt* wird die oben beschriebene Dualgraph-Konstruktion durchgeführt.

2. Im *zweiten Schritt* werden zahlreiche Zerlegungen des gegebenen Netzes in jeweils k Segmente durchgeführt. Dafür können prinzipiell ganz unterschiedliche Segmentierungsverfahren genutzt werden. Einzige Bedingung ist, dass sie randomisierbar sind. Folgende Randomisierungsstrategien werden in [GF08] genannt, die auch beliebig kombinierbar sind:
   - Es wird zufällig gewählt, welches Segmentierungsverfahren angewendet wird.
   - Die Werte spezieller Parameter können randomisiert bestimmt werden. Ein Beispiel dafür ist die Anzahl der zu ermittelnden Segmente.
   - Zufallsvariablen können für Entscheidungen während der Berechnung einer Segmentierung zum Einsatz kommen. Dieses bietet sich beispielsweise bei der Festlegung einer konkreten Bearbeitungsreihenfolge an.
   - Zufällige Variationen am zugrunde liegenden Dualagraphen stellen eine weitere Möglichkeit dar. Beispielsweise kann eine zufallbasierte Einflussnahme auf die Kantengewichte erfolgen.

   Jede Segmentierung wird mit einem Gewichtswert versehen, der die Qualität dieser Segmentierung beschreibt.

3. Basierend auf einer Menge $\hat{S}$ von Segmentierungen wird im *dritten Schritt* für jede Netzkante der Wert einer so genannten Partitionierungsfunktion berechnet. Dabei werden für jede Kante e die entsprechenden Gewichte aller Segmentierungen aus $\hat{S}$, bei denen eine Segmentgrenze durch e verläuft, addiert.

4. Im *vierten Schritt* wird mit Hilfe der Partitionierungsfunktion für jede erzeugte Segmentierung ein Konsistenzwert bestimmt. Die Segmentierungen mit den höchsten Konsistenzwerten werden die *Most Consistent Cuts* genannt. Sie zeichnen sich dadurch aus, dass sie mit den anderen Segmentierungen im Durchschnitt am besten übereinstimmen.

Obwohl beim Randomized-Cuts-Verfahren durchaus unterschiedliche Segmentierungs-Verfahren zum Einsatz kommen können, haben Golovinskiy und Funkhouser in [GF08] nur das Min-Cut-Verfahren genutzt, bei dem die Segmentierungsberechnung auf die Berechnung eines optimalen Flußes in einem kantengewichteten Graphen zurückgeführt und mit dem Algorithmus von Edmonds und Karp gelöst wird. Um zu erreichen, dass alle Segmente in etwa die gleiche Größe haben, modifizieren die Autoren den in [KT03] angegebenen Minimum-Cut-Algorithmus derart, dass die Kantengewichte über eine Penalty-Funktion, die proportional zum Abstand vom nähesten Seed fällt, angepasst werden.

Vor der Anwendung des eigentlichen Randomized-Cuts-Verfahrens wurde in [GF08] stets ein Remeshing durchgeführt, durch welches die Polygonnetze auf eine Größe von rund 4000 Dreiecke reduziert wurden. Für ein solches reduziertes Modell beträgt die durchschnittliche Rechenzeit, die zur Ermittlung einer stabilen Partitionierungsfunktion benötigt wird, bei Verwendung eines Rechners mit einer 2.2 GHz-CPU und 2 GB Hauptspeicher bei 0.6 Sekunden pro Iteration rund 4 Minuten [GF08].

„Normalized Cuts“-Segmentierung

Hierarchisches Clustern wird als ein mögliches Vorgehen zur Bestimmung einer einzelnen Segmentierung innerhalb des Randomized-Cuts-Verfahren genannt. Das Normalized-Cuts-Verfahren [GF08] gehört zu der Klasse von Segmentierungsverfahren, die auf hierarchischem Clustern basieren. Dabei wird zunächst jede Dreiecksfläche eines Dreiecksnetzes als eigenes Segment angesehen. Anschließend findet so lange eine Vereinigung einzelner Segmente statt, bis eine von Anwenderseite vorgegebene Anzahl an Segmenten erreicht ist. Dafür werden Normalized-Cut-Kosten verwendet, die Konkavitätsinformationen enthalten. Für jedes Paar von adjazenten Segmenten werden die Cut-Kosten des Segments bestimmt, das sich durch deren Vereinigung ergibt. Diese Cut-Kosten stehen für den Aufwand, der notwendig ist, um das zusammengesetzte Segment wiederum in die beiden Teile zu zerlegen. Sie sind bei konkaven Segmentgrenzen klein, während sie einen großen Wert haben, wenn die Grenze der beiden Teil-Segmente in einer konvexen Region liegt. Für ein gegebenes Segment werden die Cut-Kosten zu allen adjazenten Segmenten aufaddiert und durch den Flächeninhalt des Segments dividiert. Der Unterschied zu den „traditionellen“ Cut-Kosten besteht darin, dass hier durch den Flächeninhalt des Segments dividiert wird, während bei „traditionellen“ Cut-Kosten durch alle Kantenkosten, die sich im Inneren des Segments befinden, geteilt wird. Die Division durch die Segment-Fläche wird durchgeführt, um eine Abhängigkeit von der Vernetzung zu vermeiden, so dass üblicherweise Segmente von ähnlicher Größe generiert werden (vgl. auch [BAT11]). Basierend auf den Normalized-Cut-Kosten wird eine Funktion definiert, die für jedes Paar von Segmenten die Wahrscheinlichkeit dafür angibt, dass dieses Paar zur Vereinigung selektiert wird. Dies geschieht iteriert, bis die gewünschte Anzahl an Segmenten erreicht ist.

„Shape Diameter“-Segmentierung

Shapira et al. stellen in [SSCO08] ein hierarchisches Segmentierungsverfahren vor, das invariant gegenüber Veränderung der Pose des Modells ist. Es basiert auf der Bestimmung des Modell-Durchmessers an jeder einzelnen Facette des Dreiecksnetzes. Bei „glatten“ Oberflächen lässt sich der Durchmesser an einem Oberflächenpunkt $\mathbf{p}$ bestimmen, indem der Abstand von $\mathbf{p}$ zum gegenüberliegenden Punkt ermittelt wird, d.h. zu dem nächsten Oberflächenpunkt, der in Richtung eines nach innen zeigenden Normalenvektors in $\mathbf{p}$ liegt. Dies kann jedoch bei Dreiecksnetzen zu Problemen führen. Deswegen definieren Shapira et al. in [SSCO08] eine *Shape-Diameter-Function* (*SDF*) auf der Modell-Oberfläche und werten diese an den Schwerpunkten der Netzdreiecke

aus. Zur Bestimmung des Wertes der Shape-Diameter-Function an einem Oberflächen-Punkt **p** wird in **p** ein Kegel mit einem vorgegebenen Öffnungswinkel um einen nach innen gerichteten Normalenvektor der Facette, auf der **p** liegt, konstruiert. Dann werden 30 Strahlen zufällig konstruiert, die innerhalb des (in Richtung des Normalenvektors unendlich ausgedehnten) Kegels liegen. Der Öffnungswinkel des Kegels sollte weder zu klein noch zu groß sein; Shapira et al. geben an, dass sich ein Öffnungswinkel von 120 Grad als gut herausgestellt hat. Für jeden Strahl wird der dem Punkt **p** am nächsten gelegene Schnittpunkt des Strahls mit dem Dreiecksnetz bestimmt. Der Einfachheit halber wird in der vorliegenden Dissertation in diesem Zusammenhang entsprechend dem in [SSCO08] gewählten Begriff „ray length“ von *Strahllänge* gesprochen, auch wenn ein Strahl im mathematischen Sinne eigentlich unendlich lang ist. Der SDF-Wert ergibt sich dann durch eine geeignete Mittelung der Strahllängen. Es werden jedoch nur die Strahlen berücksichtigt, deren Länge innerhalb einer Standardabweichung vom Median der Längen liegt. Die Mittelung der Strahllängen beinhaltet allerdings eine Gewichtung: Je größer der Winkel zwischen einem Strahl und der durch den Normalenvektor der Facette definierten Achse des Kegels ist, umso geringer ist der Anteil, den dieser Strahl am Ergebnis hat. Um zu erreichen, dass die Shape-Diameter-Function auch an den Gelenkstellen des Modells invariant von dem konkreten Winkel ist, wird eine anisotropische Glättung der SDF angewendet.

Die Bestimmung der Segmentierung erfolgt in drei Schritten. Zunächst wird für jede Dreiecksfacette der SDF-Wert am Schwerpunkt ermittelt. Aus diesen Werten können Iso-Linien auf der Oberfläche hergeleitet werden. Iso-Linien bestimmter Werte bieten sich offensichtlich als Grenzen zwischen den einzelnen Segmenten an. Realisiert wird die Berechnung solcher Linien, indem im zweiten Schritt ein Histogramm für die SDF-Werte bestimmt wird und in dieses mit Hilfe der Gaußschen Mischverteilung k Gauß-Kurven gelegt werden. Diese Kurven ermöglichen für jede Facette kanonisch die Definition eines Vektors mit k Elementen, der angibt, mit welcher Wahrscheinlichkeit die Facette zu welchem der k Segmente gehört. Daraus lässt sich recht leicht eine Segmentierung herleiten, deren Grenzen allerdings in zahlreichen Fällen noch nicht glatt und „natürlich“ sind. Um dies zu erreichen, wird im dritten Schritt ein Graph-Cut-Verfahren angewendet. Dieses berücksichtigt sowohl die oben genannten Wahrscheinlichkeitsvektoren als auch einen Energie-Term für die Glattheit und Konkavität der Segmentgrenzen. Segmentierungen unterschiedlicher Hierarchiestufen ergeben sich, indem die Anzahl k der Segmente variiert wird.

Das Shape-Diameter-Function-Verfahren ermöglicht konsistente Segmentierungen von ähnlichen Modellen sowie von verschiedenen Modellen desselben Objekts in unterschiedlichen Posen. Als schwierig haben sich jedoch Modelle herausgestellt, deren Komponenten stark von einer Zylinder-Form abweichen (vgl. [SSCO08]).

„Core Extraction“-Segmentierung

Katz et al. stellen in [KLT05] mit dem Core-Extraction-Verfahren ein auf der Extraktion der zentralen Komponente des Modells basierendes Verfahren vor. Es ist invariant

gegenüber der Pose. So soll das Core-Extraction-Verfahren beispielsweise bei einem Menschenmodell ermöglichen, dass die Arme als eigenständige Segmente erkannt werden, und zwar unabhängig davon, ob sie ausgestreckt oder angewinkelt sind oder nur nach unten hängen. Darüber hinaus soll das Core-Extraction-Verfahren bei semantisch ähnlichen Modellen, deren korrespondierende Komponenten allerdings unterschiedliche Proportionen haben, jeweils gleichartige Segmentierungen liefern, was nach [KLT05] bei anderen Verfahren durchaus nicht immer der Fall ist.

Das Core-Extraction-Verfahren erzeugt eine Segmentierungshierarchie in Form eines (nicht zwangsläufig binären) Baumes. Jeder Knoten des Baumes steht für ein Segment, die Kind-Knoten eines Knoten v_{parent} für Segmente, die eine Zerlegung des zu v_{parent} gehörenden Segments darstellen. Das Verfahren besteht aus vier zentralen Schritten (vgl. auch [CGF09]), die für jeden Knoten des Baumes erneut auszuführen sind, solange kein Abbruchkriterium erfüllt ist:

1. Im *ersten Schritt* wird das Dreiecksnetz vergröbert. Auf diese Weise wird einerseits die Berechnung beschleunigt und andererseits der Einfluss von Rauschen verringert.

2. Im *zweiten Schritt* wird das Modell durch eine Transfomation mit Hilfe der *Multi-Dimensionalen-Skalierung* (*MDS*) [Hor69] in eine Form gebracht, die unabhängig von der Pose ist. Anschaulich gesprochen werden dabei sämtliche Extremitäten auf eine Weise lang gezogen, so dass sie möglichst weit voneinander wegzeigen.

3. Auf dem durch die Transformation entstandenen Dreiecksnetz werden im *dritten Schritt* bedeutungsvolle Punkte, die so genannten *Feature Points*, ermittelt. Dafür kommen alle Eckpunkte des Netzes in Frage, für die der aufsummierte Abstand zu allen anderen Netzpunkten größer ist als der aufsummierte Abstand an sämtlichen benachbarten Punkten. Von diesen werden allerdings nur diejenigen genommen, die auf der konvexen Hülle des transformierten Netzes liegen.

4. Im *vierten Schritt* wird zunächst die zentrale Komponente des aktuellen Netzes extrahiert. Dies geschieht mittels eines *Spherical Mirroring*, bei dem eine Bounding-Kugel, die nicht zwangsläufig minimal zu sein braucht, um das Dreiecksnetz gelegt wird. Eine Spiegelung des Netzes an dieser Kugel bewirkt, dass außen liegende Teile weiter innen und innen liegende Teile weiter außen angesiedelt werden. Nachdem die konvexe Hülle der gespiegelten Punkte berechnet wurde, werden die Netzelemente, die durch die Spiegelung auf dieser konvexen Hülle landen, dem Segment zugeordnet, das die zentrale Komponente darstellt. Alle anderen Segmente ergeben sich dann kanonisch durch Entfernen dieses Segments. Die Segmentierung wird dann auf das ursprüngliche, feinere Dreiecksnetz übertragen.

Die weitere Zerlegung eines Segments wird spätestens dann abgebrochen, wenn es keine Feature Points Points mehr enthält.

Das beschriebene Vorgehen kann zu groben, ungenauen Segmentgrenzen führen. Aus diesem Grund findet auch eine Verfeinerung der Grenzen (*Cut Refinement*) statt. Dazu wird für eine begrenzte Umgebung um eine Segmentgrenze ein Flussgraph konstruiert und auf diesem ein minimaler Schnitt ermittelt. In die Kapazitätsfunktion für die Kanten fließen sowohl die Kantenlänge im Dreiecksnetz als auch Informationen über die Konvexität ein.

Das Core-Extraction-Verfahren ist in der hier beschriebenen Form nicht gut für die Segmentierung von CAD-Modellen geeignet, da die MDS-Repräsentation dazu führen kann, dass keine sinnvolle Segmentierung möglich ist. Aus diesem Grund empfehlen Katz et al. in [KLT05], bei CAD-Modellen auf die MDS-Transformation zu verzichten. Ein weiterer Nachteil zeigt sich bei Modellen, die ringförmige Strukturen enthalten (vgl. [KLT05]). Beispiele dafür sind eine Tasse mit Henkel oder ein Tier, dessen Schwanzspitze den Rumpf wieder berührt. Bei solchen ringförmigen Strukturen können die berechneten Segmentgrenzen stark von den erwarteten abweichen.

„Random Walk“-Segmentierung

Das Random-Walk-Verfahren von Lai et al. [LHMR08] stellt einen weiteren Ansatz dar, der auf Seed-Punkten basiert. Ein Seed-Punkt entspricht dabei der Facette des Dreiecksnetzes, auf der er angeordnet wird. Die Autoren empfehlen jedoch, bei großen Netzen zunächst die Anzahl der Facetten zu reduzieren, so dass ca. 10 000 bis maximal 20 000 übrig bleiben.

Für jede der drei Kanten einer jeden Dreiecksfacette wird die Wahrscheinlichkeit dafür definiert, dass dieses Dreieck bei einem Random-Walk über diese Kante wieder verlassen wird, wobei die Summe der Wahrscheinlichkeiten für die drei Kanten stets 1 ergibt. Dabei versteht man unter einem *Random-Walk* ein zufälliges Ablaufen des Netzes entsprechend der Dualgraph-Kanten ausgehend von einer vorgegebenen Start-Facette unter Berücksichtigung der oben genannten Übergangswahrscheinlichkeiten bei der Entscheidung, wie weiterzulaufen ist. Eine Dreiecksfacette t wird dem Seed t_{seed} zugeordnet, für den die Wahrscheinlichkeit am größten ist, dass ein von t aus startender Random-Walk t_{seed} eher erreicht als alle anderen Seeds. Da nach der Minima-Rule [HR84] konkave Objektregionen gute Kandidaten für Segmentgrenzen sind, definieren Lai et al. die Übergangswahrscheinlichkeiten unter Berücksichtigung des dihedralen Winkels zwischen den beiden jeweiligen benachbarten Facetten. Bei natürlichen Modellen wie zum Beispiel Modellen von Tieren gewichten sie allerdings konkave Winkel deutlich stärker als konvexe, während bei technischen Modellen beide als gleichwichtig angesehen und damit auch gleich gewichtet werden. Ein weiterer Unterschied besteht darin, dass bei technischen Modellen zusätzlich zu den dihedralen Winkeln auch Abwandlungen der Gaußschen Krümmung und der Hauptkrümmung zur Berechnung der Übergangswahrscheinlichkeiten herangezogen werden.

Lai et al. geben in [LHMR08] sowohl eine interaktive als auch eine automatische Variante des Random-Walk-Verfahrens an. Bei der *interaktiven Variante* platzieren die Anwendenden für jedes erwartete Segment einen Seed auf dem dreidimensionalen

Modell. Die anschließende Segment-Berechnung beinhaltet keine weiteren interaktiven Eingriffsmöglichkeiten. Das manuelle Setzen der Seed-Punkte entfällt bei der *automatischen Variante*. Zur automatischen Seed-Anordnung geben Lai et al. mit dem Coarse-Scale-Seeding und dem Fine-Scale-Seeding zwei Möglichkeiten an. Beim *Coarse-Scale-Seeding* wird der erste Seed-Punkt möglichst weit weg vom Schwerpunkt des Dreiecksnetzes angeordnet. Alle weiteren Seeds werden nacheinander derart angeordnet, dass der geodätische Abstand zu allen anderen maximiert wird. Es werden so lange neue Seeds hinzugefügt, bis sich dieser Abstand im Vergleich zum vorher eingeführten signifikant verringert.

Das Coarse-Scale-Seeding eignet sich eher zur Segmentierung großer Komponenten als feingranularer Strukturen. Für letztere haben Lai et al. ein *Fine-Scale-Seeding* vorgeschlagen, bei dem zunächst mehr Seed-Punkte verteilt werden als Segmente erwünscht sind[2]. Daraus resultiert eine Übersegmentierung. In einem zweiten Schritt werden die Segmente dann solange hierarchisch vereinigt, bis nur noch die gewünschte Anzahl an Segmenten übrig bleibt. Die automatische Variante mit dem Fine-Scale-Seeding ist bei der Benchmark-Auswertung von Chen et al. [CGF09] zum Einsatz gekommen.

„Fitting Primitives“-Segmentierung

Attene et al. haben im Jahr 2006 ein Segmentierungsverfahren für den so genannten Reverse-Engineering-Prozess vorgestellt [AFS06], bei dem ein Objekt in einzelne Segmente zerlegt wird, die sich möglichst gut durch geometrische Grundformen wie beispielsweise Kugeln oder Zylinder approximieren lassen. Die Grundformen werden *Fitting Primitives* genannt. In erster Linie eignet sich das Verfahren für technische Modelle, beispielsweise aus dem CAD-Umfeld. Vorgegangen wird nach dem Bottom-Up-Prinzip, d.h. am Anfang bildet jede einzelne Facette des Dreiecksnetzes ein eigenes, einelementiges Cluster. Ausgehend von dieser Situation werden sukzessive benachbarte Cluster zu einem neuen zusammengezogen, so dass sich eine hierarchische, binäre Segmentierung[3] ergibt. Dies geschieht auf dem Dualgraphen, dessen Kanten mit Gewichten versehen sind, welche die Kosten für das Zusammenziehen benachbarter Cluster angeben. Es werden jeweils die beiden Knoten des Dualgraphen zu einem neuen, das zusammengezogene Cluster symbolisierenden Knoten v' zusammengezogen, die über die Kante mit den niedrigsten Kosten verbunden sind. Nach einer solchen Vereinigung zweier Cluster werden die Kosten für alle zu v' inzidente Kanten aktualisiert. Die Kosten sind so konstruiert, dass sie angeben, wie gut das Cluster, das sich durch das Zusammenziehen der beiden Endknoten der jeweiligen Dualgraph-Kante ergibt, durch eine der für den Segmentierungsvorgang zulässige geometrischen Grundformen approximiert werden kann. Je besser eine solche Approximation ist, umso geringer sind die

[2] Im Unterschied zum Coarse-Scale-Seeding schlagen Lai et al. vor, die Seeds in diesem Fall annähernd gleichmäßig über das Netz verteilt werden, wobei Seeds jedoch eher in der Umgebung besonderer Feature-Punkte bzw. auf herausstehenden Teilen angeordnet werden, als an anderen Stellen [LHMR08].

[3] Da stets zwei Cluster vereinigt werden, resultiert ein binärer Baum, dessen Knoten für die einzelnen Cluster stehen.

Kosten. Die Autoren haben in [AFS06] Kugeln, Zylinder und Ebenen verwendet. Das Hinzufügen weiterer Arten von Fitting Primitives ist ebenfalls möglich.

„K-Means Clustering“-Segmentierung

Das *K-Means-Clustering*-Verfahren zur Segmentierung von Dreiecksnetzen basiert auf Seed-Punkten. Es ist von Shlafman et al. in [STK02] vorgestellt worden und geht iteriert vor. Zunächst werden die Facetten des gegebenen Dreiecksnetzes jeweils dem ihnen am nächsten liegenden Seed zugeordnet, woraus sich Patches ergeben. Die Seeds sind jedoch nicht zwangsläufig optimal angeordnet. Deswegen findet nach der Patch-Ermittlung eine Verschiebung der Seeds entsprechend dem Vorgehen statt, das aus dem klassischen K-Means-Clustering bei Mengen bekannt ist: Für jedes Patch wird die Facette ermittelt, deren Summe der Abstände zu allen anderen Facetten des Patches minimal ist. Diese Facette liegt in gewissem Sinne zentral in dem Patch und wird deswegen dann als neue Seed-Position genommen. Anschließend findet ausgehend von dem sich so ergebenden neuen Seeds eine erneute Patch-Berechnung statt. Der Prozess der Patch-Berechnung mit anschließender Verschiebung der Seeds wird so lange iteriert, bis eine Konvergenz eintritt.

Chen et al. haben das K-Means-Clustering-Verfahren leicht variiert [CGF09]. Sie berücksichtigen bei der Distanzberechnung auf dem Dualgraph die Winkel zwischen Dreiecken, die zu dem Dualgraph-Pfad korrespondieren, nicht in der Originalform, sondern entsprechend der Beschreibung in [FKS+04]. Die Winkel werden in Strafkosten überführt, welche die Distanz entsprechend vergrößern. Diese modifizierte Variante des K-Means-Clustering-Verfahrens liegt auch den Benchmark-Ergebnissen zugrunde, die in Kapitel 8 angegeben sind.

„Labeling and Learning“-Segmentierung

Kalogerakis et al. stellen in [KHS10] einen Ansatz vor, der sich dahingehend von den anderen in diesem Kapitel genannten Verfahren unterscheidet, dass der Bestimmung der Segmentierung zunächst ein Lernprozess anhand mehrerer anderer Modelle vorausgeht, die jeweils eine ähnliche Semantik haben wie das zu segmentierende Modell. Dabei werden nicht nur die Segmente erzeugt, sondern auch ein so genanntes *Labeling* der Segmente unter Verwendung eines *Conditional Random Field* (*CRF*) durchgeführt. Die Label geben dabei an, um welches semantische Teil es sich handelt. Beispiele für Label bei einem Menschenmodell sind demnach „Arm“ oder „Bein“.

Jede Facette besitzt einen *Feature-Vector*, der unter anderem Aspekte aus der Hauptkomponentenanalyse und der Shape-Diameter-Function sowie Distanzen zu Medialpunkten enthält. Zu jeder Netzkante wird ein *Pairwise-Feature-Vector* gespeichert, der beispielsweise den dihedralen Winkel sowie Krümmungen und diverse Differenzen von Feature-Werten der beiden Facetten enthält.

Die Label werden pro Facette bestimmt. Dazu ist eine Zielfunktion zu minimieren, die einerseits für jede Facette die Konsistenz ihres Feature-Vector mit dem Label in Form

eines *Unary-Energy-Terms* misst, und andererseits die Konsistenz der Label benachbarter Facetten mit dem zur entsprechenden Kante gehörenden Pairwise-Feature-Vector in Form eines *Pairwise-Energy-Terms*. Die Optimierung der Zielfunktion erfolgt durch Anwendung eines *Alpha-Expansion-Graph-Cut-Verfahrens*.

Damit eine gute Segmentierung erzielt werden kann, werden die CRF-Parameter mit Hilfe einer Menge von Trainingsmodellen erlernt, die bereits mit Labeln versehen sind. Auf Basis der gegebenen Label werden geeignete Parameter für zwei Joint-Boost-Classifier ermittelt, die im Unary-Energy-Term bzw. im Pairwise-Energy-Term verwendet werden. Darüber hinaus werden mit den Trainingsmodellen weitere im Pairwise-Energy-Term enthaltene Parameter bestimmt.

Mit dem Labeling-and-Learning-Verfahren lassen sich recht gute Segmentierungsergebnisse erzielen. Allerdings werden stets geeignete und mit Labeln versehene Trainingsmodelle benötigt. Dies ist als ein klarer Nachteil dieses Verfahrens gegenüber anderen Ansätzen anzusehen, denen keine Lernphase vorausgeht, zumal das Erlernen der Parameter auf einem 2.66 GHz-Prozessor durchaus mehrere Stunden an Zeit in Anspruch nehmen kann (vgl. [KHS10]). Auch der in der vorliegenden Dissertation vorgestellte Segmentierungsansatz unterscheidet sich von dem Labeling-and-Learning-Ansatz grundlegend in der Tatsache, dass er keine Lernphase beinhaltet. Der Verzicht auf eine Lernphase ermöglicht es, für Modelle, zu denen es noch keine adäquaten Trainingsmodelle gibt, gute Segmentierungen zu ermitteln.

„Semi-Supervised-Weighted-Graph“-Segmentierung

In [BAT11] stellen Bergamasco et al. einen semi-überwachten Ansatz vor, bei dem zunächst Seed-Punkte manuell zu setzen sind. Bewusst haben sie sich gegen ein vollautomatisches Verfahren entschieden, da es häufig mehrere konkurrierende Zerlegungen eines Modells in Segmente gibt, die alle gleich sinnvoll sind. Durch das obligatorische Platzieren von Seed-Punkten haben die Anwendenden die Möglichkeit, das Verfahren dahingehend zu beeinflussen, dass die Zerlegung des Modells entsprechend ihrer semantischen Auffassung des Modells erfolgt.

Wie schon bei einigen anderen Segmentierungsverfahren, kommt auch beim Semi-Supervised-Verfahren ein gewichteter Dualgraph zur Berechnung der Segmente zum Einsatz. Das Gewicht einer Kante gibt an, wie groß der Aufwand ist, die zugehörige Kante des Dreiecksnetzes zu überschreiten. Je größer der Winkel zwischen zwei benachbarten Dreiecken ist, umso größer ist das Gewicht. Des Weiteren fällt das Gewicht mit steigendem Abstand der Schwerpunkte der beiden Dreiecke voneinander. Für die Dualgraph-Kante zwischen zwei Dualgraph-Knoten i und j wird das Gewicht $w(i,j)$ nach

$$w(i,j) := \frac{1 - \langle \vec{n}_i, \vec{n}_j \rangle}{\|\vec{p}_i - \vec{p}_j\|} \tag{3.1}$$

berechnet, wobei $\vec{p}_i$ und $\vec{p}_j$ die Stützvektoren zu den beiden Dreiecksschwerpunkten und $\vec{n}_i$ sowie $\vec{n}_j$ Normalenvektoren der Dreiecke sind. Bei dieser Gewichtsdefinition

wird allerdings nicht wie beispielsweise in [LHMR08] oder [GF08] zwischen konvexen und konkaven Winkeln unterschieden.

In jedem Schritt wird das Modell binär, d.h. in genau zwei Segmente zerlegt[4]. Dazu platzieren die Anwendenden einige Seed-Punkte und unterteilen die zugehörigen Knoten in „grüne“ und „rote“ Knoten, wobei sowohl für das „grüne“ Segment als auch für das „rote“ deutlich mehr als nur ein Seed gesetzt werden können. Ähnlich einer parallelen Breitensuche werden ausgehend von diesen gefärbten Knoten unter Berücksichtigung der im Vorfeld berechneten Kantengewichte die noch nicht gefärbten Knoten dem roten bzw. dem grünen Segment zugeordnet. Dabei erhält ein Knoten die Farbe des Knotens, von dem aus er erreicht worden ist. Eine hierarchische Segmentierung erhält man, indem iteriert segmentiert wird. Dies bedeutet jedoch auch, dass bei jedem Iterationsschritt erneut Benutzeraktionen notwendig sind.

[4]Genau genommen wird das Modell lediglich in ein wirkliches Segment und den „restlichen Modellteil“ zerlegt. Letzteres wird anschließend weiter zerlegt.

Kapitel 4

Überblick über das EvOpSeg-Verfahren

Das in den folgenden Kapiteln dieser Arbeit vorgestellte Segmentierungsverfahren wird EvOpSeg-Verfahren genannt. Wie schon erwähnt steht EvOpSeg für ***Ev****olutionary* ***Op****timized Mesh* ***Seg****mentation*. Bei der Beschreibung des EvOpSeg-Verfahrens ist eine genaue sprachliche Abgrenzung unterschiedlicher Teilnetz-Arten erforderlich: Berechnete Teilnetze, die nicht unbedingt bedeutungsvolle Teile darstellen, werden als *Patches* bezeichnet[1], wohingegen Teilnetze, die zumindest annähernd mit so genannten bedeutungsvollen Objektteilen übereinstimmen, *Segmente* genannt werden. Für logische semantische Teile eines Objekts, also solche, die ein Mensch aufgrund seines Wissens über das Objekt als semantische Teile wahrnimmt, wird der Begriff *Komponente* verwendet.

Die Zerlegung eines Dreiecksnetzes in Segmente erfolgt beim EvOpSeg-Verfahren ausgehend von Seed-Punkten. Die Ränder zwischen den einzelnen Segmenten müssen gewisse Segmentierungskriterien erfüllen. Insbesondere besagen die beim EvOpSeg-Ansatz gewählten Kriterien, dass die Segmenträndern im Wesentlichen durch konkave Objektregionen verlaufen sollen und dass sie möglichst glatt zu sein haben. Das erste Kriterium basiert auf der so genannten Minima-Rule [HR84], dem zweiten liegt die Annahme zugrunde, dass Grenzen zwischen Komponenten, also den semantischen Teilen eines Objekts, üblicherweise nicht „ausgefranst“ sind. Diese Kriterien eigenen sich sowohl für natürliche als auch für technische Objekte.

EvOpSeg stellt ein mehrstufiges Segmentierungsverfahren dar. Es beinhaltet im Wesentlichen drei zentrale Bestandteile:

1. **Ermittlung der Seed-Punkte:** Die Bestimmung geeigneter Seed-Punkte erfolgt zweistufig, wobei allerdings der zweite Schritt im Gegensatz zu dem in [EM12] beschriebenen Verfahren optional ist: Im ersten Schritt werden initiale Seed-Punkte, die so genannten *Initial-Seeds*, derart gesetzt, dass die Patch-Berechnung (vgl. Aufzählungspunkt 2.) eine Zerlegung des Dreiecksnetzes liefert,

[1] In diesem Zusammenhang muss ein Patch jedoch nicht zwangsläufig topologisch äquivalent zu einer Kreisscheibe sein.

die das erwartete Ergebnis zumindest grob approximiert.
Zusätzliche Seed-Punkte, die so genannten *Satelliten-Seeds*, werden im optionalen zweiten Schritt erzeugt. Jeder initiale Seed-Punkt bildet zusammen mit einigen Satelliten-Seeds ein Seed-Cluster. Dieses ermöglicht eine bessere Einflussnahme auf die Segmenträndern, da das Bewegen eines Satelliten-Seeds oder ein Variieren seines Gewichts in der Regel nur einen Abschnitt des Patch-Randes beeinflusst.

2. **Patch-Ermittlung:** Ausgehend von einer endlichen Menge gewichteter Seed-Punkte wird eine Zerlegung des Oberflächennetzes in einzelne Patches ermittelt. Dies läuft ähnlich ab wie bei gewichteten Voronoi-Diagrammen [AE84]: Einem Seed-Punkt $\mathbf{p}$ werden alle Dreiecke zugeordnet, deren Mittelpunkte näher an $\mathbf{p}$ liegen als an allen anderen Seed-Punkten. Dafür ist ein geeignetes Distanzmaß zu wählen, das die für die Segmentierung relevanten Oberflächeneigenschaften berücksichtigt. In dieser Arbeit kommen zwei solcher Distanzmaße zum Einsatz: eine *winkelbasierte Distanz* sowie eine zweite, welche die winkelbasierte Distanz um krümmungsbasierte Merkmale erweitert. Die Beschaffenheit der beiden Maße bewirkt eine Minimierung des Auftretens von Patchrändern, die nicht in Regionen liegen, welche dem Segmentierungskriterium genügen.

3. **Optimierung:** Häufig stimmen Patches nicht exakt mit bedeutungsvollen Teilen des Objekts, d.h. mit den Komponenten, überein. Aus diesem Grund wird in dieser Arbeit eine Patch-Optimierung mit einer Evolutionsstrategie durchgeführt [Rec73, Sch75], die unter anderem geeignete Positionen sowie Gewichte für die einzelnen Seed-Punkte ermittelt. Die Zielfunktion bewertet dabei die Patchränder unter Berücksichtigung der Segmentierungskriterien.

Abbildung 4.1 symbolisiert die genannten Schritte. Zunächst werden Seed-Punkte definiert, die dort gelb markiert sind. Im vorliegenden Fall handelt es sich um Seed-Cluster, die sich durch das Hinzufügen von Satelliten-Seeds zu den Initial-Seeds ergeben haben. Das Ergebnis der von den Seeds ausgehenden Patch-Ermittlung braucht nicht zwangsläufig den Erwartungen zu entsprechen, was im mittleren Teilbild am Hinterlauf zu sehen ist. Die anschließende Optimierung bewirkt, dass gute Grenzen zwischen den Patches entstehen.

Diese einzelnen Bestandteile des EvOpSeg-Verfahrens werden in den nachfolgenden Kapiteln 5 bis 7 behandelt. Kapitel 5 befasst sich mit der Berechnung einer so genannten *Initialsegmentierung*. Dazu werden zunächst Initial-Seeds, also „normale“ Seed-Punkte ohne Berücksichtigung der Satelliten-Seeds, bestimmt. Ausgehend von diesen erfolgt die Patch-Ermittlung. Im Anschluss daran haben die Anwendenden noch die Möglichkeit, manuell einzugreifen – beispielsweise durch Gruppieren von Patches oder das Hinzufügen weiterer Seeds, wobei letzteres vom Prinzip her keinen Unterschied zum Platzieren von Initial-Seeds aufweist.

Eine Initialsegmentierung besitzt Eigenschaften, die sowohl für eine manuelle als auch für eine automatische Optimierung nachteilig sein können. So kann die Optimierung bewirken, dass zwar ein Teilstück des Patchrandes im Nachhinein die Komponentengrenze besser approximiert, gleichzeitig aber ein anderer Randabschnitt des selben Pat-

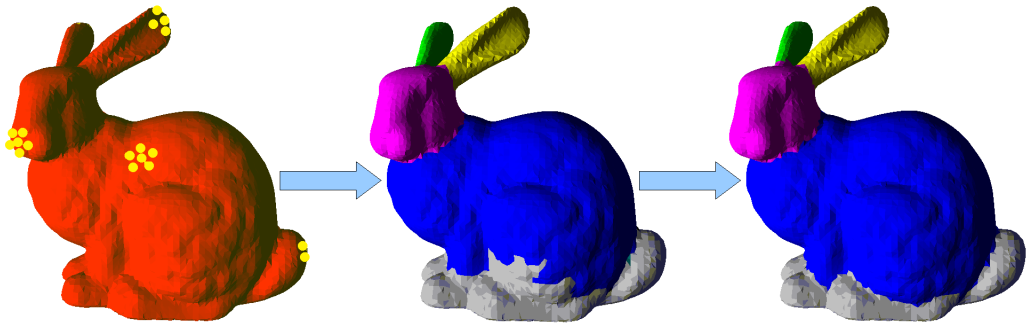


Abbildung 4.1: *Ausgehend von Seed-Punkten (links) wird eine Segmentierung durchgeführt. Die entstandenen Patches (Mitte) werden anschließend optimiert (rechts).*

ches verschlechtert wird. Abhilfe schaffen Satelliten-Seeds, die in Kapitel 6 eingeführt werden. Die Verwendung der Satelliten-Seeds ist optional. Da sie sich allerdings in vielen Situationen als hilfreich erweisen haben, wird im Folgenden davon ausgegangen, dass sie – sofern nicht anders erwähnt – standardmäßig verwendet werden. Unabhängig davon, ob nun Satelliten-Seeds zum Einsatz kommen, können die Patches mit einer Evolutionsstrategie optimiert werden. Darauf wird ausführlich in Kapitel 7 eingegangen. Üblicherweise führt die Optimierung zu einem guten Resultat, so dass der Segmentierungsprozess damit beendet ist. Sollte das Ergebnis jedoch nicht zufriedenstellend sein, besteht noch immer die Möglichkeit, manuelle Eingriffe vorzunehmen und gegebenenfalls anschließend die sich daraus ergebenden Patches erneut evolutionär zu optimieren.

Abbildung 4.2 auf der nächsten Seite gibt einen graphischen Überblick über das EvOpSeg-Verfahren und die soeben beschriebenen Abläufe. Zur Darstellung ist eine vereinfachte Form eines UML-Aktivitätsdiagramms [RHQ+05, S. 265ff] verwendet worden. Vereinfacht ist es insofern, als dass auf Entscheidungsknoten sowie Vereinigungsknoten verzichtet worden ist und stattdessen die entsprechenden Kanten direkt an den Aktionsknoten (abgerundete Vierecke) starten bzw. enden. Auch anhand dieser Abbildung wird deutlich, dass die Ermittlung der Satelliten-Seeds, manuelle Anpassungen durch die Anwendenden sowie die Optimierung der Patches optional sind. Sofern nicht anders erwähnt, wird jedoch davon ausgegangen, dass bei einem typischen Segmentierungsablauf sowohl Satelliten-Seeds verwendet als auch eine evolutionäre Patch-Optimierung durchgeführt wird.

Das im Rahmen dieser Dissertation entstandene Programm `EvOpSeg-Tool` ermöglicht die Segmentierung dreidimensionaler, als Dreiecksnetz gegebener Modelle nach dem EvOpSeg-Verfahren. Eine ausführliche Beschreibung dieses Programms ist im Anhang A.1 zu finden.

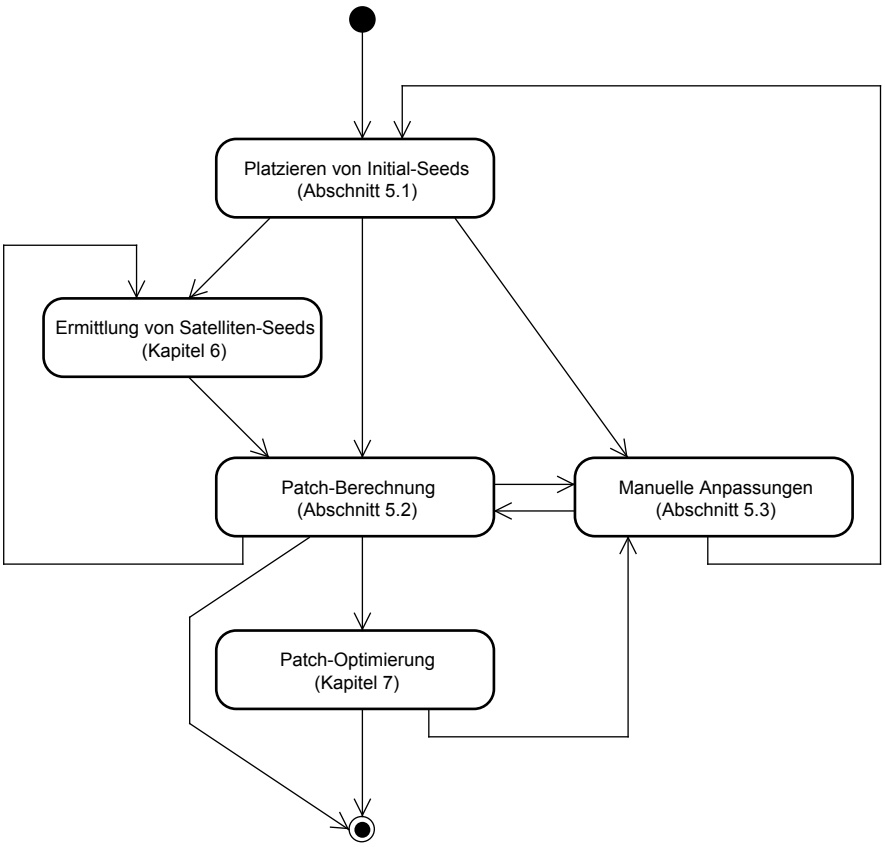


Abbildung 4.2: *Überblick über den grundsätzlichen Ablauf beim EvOpSeg-Verfahren sowie über die Kapitelstruktur.*

Kapitel 5

Initialsegmentierung

Beim EvOpSeg-Verfahren wird unter einer *Initialsegmentierung* eine Zerlegung des als Dreiecksnetz gegebenen Modells in einzelne Teile verstanden, bei denen es sich nicht zwangsläufig um bedeutungsvolle Teile zu handeln braucht. Bei Bedarf können diese Teile anschließend durch weitere Maßnahmen wie beispielsweise Anwendung einer Evolutionsstrategie optimiert werden. Sie werden im Folgenden, wie schon zuvor erwähnt, *Patches* genannt.

Seed-Punkte kommen bei einigen bekannten Segmentierungsansätzen zum Einsatz, so z.B. in [MPS+04, KJS07, LHMR08]. Dies ist ebenfalls bei der in diesem Kapitel beschriebenen Initialsegmentierung der Fall. Hier erfolgt die Berechnung der Patches ausgehend von Seed-Punkten, die in diesem Zusammenhang auch *Initial-Seeds* oder *initiale Seed-Punkte* genannt werden. Ein auf Seed-Punkten basierendes Verfahren hat die positive Eigenschaft, dass für die Definition einer ganzen Region häufig ein einziger Seed-Punkt ausreicht. Die Anzahl der Freiheitsgrade ist also verhältnismäßig gering, wodurch ermöglicht wird, dass relativ leicht und intuitiv auf die Gestalt der Regionen Einfluss genommen werden kann. Des Weiteren eignen sich Seed-Punkte sehr gut für eine anschließende evolutionäre Optimierung der Patches.

Die Ermittlung geeigneter Seed-Punkte kann sowohl manuell als auch automatisch erfolgen. Darüber hinaus ist auch ein hybrides Seeding möglich, bei dem einige Seed-Punkte manuell und andere automatisch angeordnet werden. Mit der Ermittlung geeigneter Seed-Punkte entsprechend dieser drei Seeding-Varianten befasst sich Abschnitt 5.1, bevor in Abschnitt 5.2 auf die Berechnung der Patches eingegangen wird. Eine Beschreibung interaktiver Eingriffsmöglichkeiten durch die Anwendenden wird in Abschnitt 5.3 gegeben.

5.1 Ermittlung initialer Seed-Punkte

Die Berechnung der Patches basiert beim EvOpSeg-Verfahren auf speziellen Seed-Punkten, den so genannten *gewichteten Seed-Punkten*. Bei einem gewichteten Seed-

Punkt wird über einen positiven Gewichtswert, dem so genannten *Seed-Gewicht*, dessen Einfluss gesteuert. Diese Seed-Gewichte und deren Auswirkung sind mit denen von gewichteten Voronoi-Diagrammen [AE84, SNKS09] vergleichbar, bei denen das Vergrößern eines Gewichts zu einem stärkeren Einfluss des entsprechenden Voronoi-Centers führt. Für die Anordnung der Seed-Punkte sind die Seed-Gewichte, auf die im Rahmen der Patch-Ermittlung in Abschnitt 5.2 ausführlicher eingegangen wird, nicht weiter von Interesse. Im Folgenden wird im Zusammenhang mit dem EvOpSeg-Verfahren stets vereinfachend von Seed-Punkten anstatt von gewichteten Seed-Punkten gesprochen.

Beim EvOpSeg-Verfahren können als Seed-Punkte ausschließlich Schwerpunkte der Netzfacetten gewählt werden, was die Freiheitsgrade enorm reduziert und sich bei der Optimierung der Patches (s. Kapitel 7) auszahlt. Da die Patches disjunkt sein müssen, darf jedes Dreieck des gegebenen Oberflächennetzes maximal einen Seed-Punkt enthalten. Ein Dreieck, das einen Seed-Punkt enthält, wird *Seed-Dreieck* oder *Seed* genannt.

Im einfachsten Fall besitzt jedes Patch genau einen Seed, so dass die Anzahl der Seeds mit der Patch-Anzahl übereinstimmt. Es ist aber auch möglich, dass mehrere Seeds zu demselben Patch gehören. In diesem Fall spricht man von einem *Seed-Cluster*. Sie können beispielsweise bei größeren Patches mit eher komplexen Grenzen zu anderen Komponenten nützlich sein. Dabei ist sicherzustellen, dass die Seeds eines Clusters ein *zusammenhängendes Patch* induzieren, d.h. das Patch darf nicht in getrennte Teilstücke zerfallen. Seed-Punkte können sowohl manuell als auch automatisch auf dem Dreiecksnetz angeordnet werden. Darüber hinaus ist eine Mischung aus beiden Varianten möglich, was als *hybrides Initial-Seeding* bezeichnet wird. Auf diese drei Seeding-Varianten wird in den Unterabschnitten 5.1.1 bis 5.1.3 eingegangen.

5.1.1 Manuelles Initial-Seeding

Manuell lassen sich sowohl Seeds, die jeweils ein eigenes Patch induzieren, als auch ganze Seed-Cluster mit recht geringem Interaktionsaufwand platzieren. Im Wesentlichen markieren die Anwendenden lediglich solche Dreiecke des gegebenen dreidimensionalen Modells, die Seed-Dreiecke darstellen sollen. Dazu kann das Modell im Raum gedreht sowie an dieses heran- oder von ihm weggezoomt werden. Eine Heuristik zum Platzieren von Initial-Seeds, die jeweils ein gesamtes Patch induzieren, besteht darin, für jede Komponente einen Initial-Seed anzugeben, der möglichst mittig auf der Komponente liegt. Die Motivation für diese Heuristik ist durch die in dieser Arbeit verwendete Patch-Ermittlung gegeben, die in Abschnitt 5.2 beschrieben wird und bei der einem Seed-Punkt alle Dreiecke zugeordnet werden, die bezüglich einer geeigneten Distanzfunktion näher an diesem liegen als an allen anderen Seed-Punkten.

Um ein Cluster von Initial-Seeds manuell zu erzeugen, selektieren die Anwendenden einzelne Dreiecke wie oben beschrieben und geben zusätzlich an, welche dieser Seeds zusammengehören. Oft liegen die Seeds eines Clusters relativ nahe beieinander. Je weiter sie auseinander liegen, umso größer ist die Wahrscheinlichkeit, dass das Patch in getrennte Teile zerfällt.

Eine besondere Form von Seed-Clustern sind *Seed-Linien*. Bei Seed-Linien bilden alle Seed-Dreiecke des Clusters einen – in der Regel offenen – Dreieckszug. Die manuelle Angabe solcher Dreieckszüge stellt in der Praxis keinen großen Aufwand für die Anwendenden dar. Beispielsweise können sie ähnlich wie in [FKS+04] direkt eingezeichnet werden[1]. Darüber hinaus ist es auch möglich, einzelne Dreiecke zu selektieren, die auf dem Dreieckszug liegen sollen, und diese werden entsprechend der Reihenfolge so verbunden, dass ein kürzester Dreieckszug entsteht. In Abbildung 5.1 sind normale Seeds (links), allgemeine Seed-Cluster (Mitte) und Seed-Linien (rechts) exemplarisch dargestellt. Die Seed-Dreiecke sind gelb markiert, während die Patches in anderen Farben gehalten sind. Obwohl sich die Patches bei diesem konstruierten Beispiel nur marginal unterscheiden, bedeutet dies keineswegs, dass die unterschiedlichen Seed-Arten gleichbedeutend sind. Auf die Potentiale der Seed-Cluster im Allgemeinen und der Seed-Linien im Speziellen wird im weiteren Verlauf der Arbeit noch näher eingegangen.

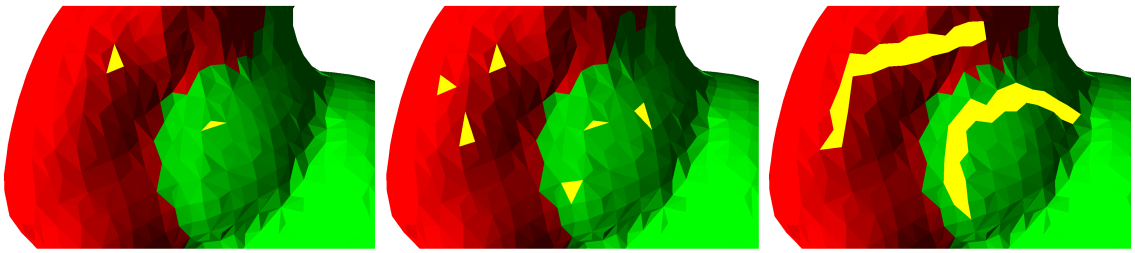


Abbildung 5.1: *Schematische Darstellung verschiedener Seed-Arten: „normale“ Seeds (links), allgemeine Seed-Cluster (Mitte) und Seed-Linien (rechts).*

5.1.2 Automatisches Initial-Seeding

Ziel der automatischen Seed-Ermittlung ist die Bestimmung einer Menge von Seed-Punkten, die für die Patch-Ermittlung geeignet sind. Die Frage, welche Dreiecke als Seeds in Betracht kommen, lässt sich nicht pauschal beantworten. Gute Kandidaten scheinen jedoch Dreiecke an exponierten Objektstellen zu sein, also beispielsweise solche, die am Ende von Extremitäten liegen. Solche Stellen können beispielsweise durch Multidimensionale Skalierung (MDS) [Hor69, SS08, SNKS09] ermittelt werden, was unter anderem auch in [KLT05] zur Ermittlung von Feature Points genutzt wird; anschaulich gesprochen werden dabei alle Extremitäten des Modells gestreckt und so ausgerichtet, dass sie möglichst weit voneinander weg zeigen. Exponierte Stellen alleine reichen jedoch bei einem Seed-basierten Ansatz nicht aus; auch „zentrale Komponenten“ wie der Körper eines Menschenmodells oder aber ein Oberschenkel müssen mit Seeds versehen werden können. Aus Granularitätsgesichtspunkten ist es auch nicht ratsam, zunächst an sämtlichen exponiert liegenden Punkten Seeds zu platzieren und erst

[1] In [FKS+04] wird auf diese Weise allerdings ein etwas breiterer Bereich definiert. Eine Übertragung dieser „Maltechnik“ auf Dreieckszüge sollte aber kanonisch möglich sein.

danach innen liegende Komponenten damit zu versehen, da auf diese Weise davon auszugehen ist, dass außen liegende Komponenten äußerst feingranular segmentiert werden während gleichzeitig innen liegende nur recht grob segmentiert sind. Eine Möglichkeit, dies zu vermeiden und zudem auch innen liegende Komponenten mit Seed zu versehen, besteht darin, die Seeds gleichmäßig über das Objekt zu verteilen. Diese Idee liegt dem hier vorgestellten Ansatz zum automatischen Seeding zugrunde, der dem *Coarse-Scale-Seeding* aus [LHMR08] sowie dem Vorgehen beim Initial Center Placement aus [SS08, SNKS09] ähnelt. Anders als beim Coarse-Scale-Seeding ist die Anzahl der gewünschten Patches im vorliegenden Fall jedoch von den Anwendenden vorzugeben, denen eine Abschätzung der Anzahl aufgrund ihres semantischen Wissens über das Objekt üblicherweise sehr leicht fällt. Insbesondere wissen sie besser, welche Segmentierungsgranularität erwartet wird, als es ein Algorithmus abschätzen könnte. Von dem Initial Center Placement unterscheidet sich das hier beschriebene automatische Seeding darin, wie die ersten beiden Seeds ermittelt werden.

Zunächst wird die Facette t_0 des gegebenen Dreiecksnetzes bestimmt, die von dem Modellschwerpunkt[2] den geringsten euklidischen Abstand im Raum hat. Diese wird jedoch nicht als Seed-Dreieck genommen, da in der Regel mit besseren Segmentierungsergebnissen zu rechnen ist, wenn die ersten Seed-Punkte an den Endpunkten des Objekts bzw. der Extremitäten platziert werden. Nach Konstruktion ist davon auszugehen, dass t_0 eher mittig als an einem solchen Ende liegt. Deswegen wird das Dreieck t_1 gesucht, das von t_0 den größten geodätischen Abstand hat und als erstes Seed-Dreieck genommen. Alle weiteren Seed-Dreiecke werden sukzessive wie folgt ermittelt: Sei T_{seed} die Menge aller bereits ermittelter Seed-Dreiecke und $n = |T_{seed}|$ die Anzahl dieser Seeds. Ferner sei T_{rest} die Menge aller übrigen Facetten. Das $(n + 1)$-te Seed-Dreieck t_{new} ist dasjenige aus der Menge T_{rest}, dessen minimaler geodätischer Abstand von T_{seed} maximal ist. Damit berechnet sich t_{new} nach

$$t_{new} = \underset{t \in T_{rest}}{\operatorname{argmax}}\, d_{geo}(t, T_{seed}). \tag{5.1}$$

Das Argument des Maximums braucht allerdings nicht immer eindeutig zu sein, d.h. argmax kann – insbesondere bei stark symmetrischen Netzen bzw. Netzbereichen – durchaus eine Menge von mehreren möglichen Lösungen liefern. In einem solchen Fall kann eine beliebige (dann wäre die Seed-Dreieck-Ermittlung nichtdeterministisch) oder aber immer die zuerst bestimmte Lösung gewählt werden. Im Rahmen dieser Arbeit durchgeführte Experimente haben jedoch gezeigt, dass t_{new} normalerweise eindeutig bestimmt ist, so dass dieser Wahl keine zentrale Bedeutung zukommt.

Die Konstruktion der Menge T_{seed} bewirkt, dass alle Seed-Dreiecke gemäß der geodätischen Distanz möglichst weit auseinander liegen. Abbildung 5.2 veranschaulicht diese Aussage. Im linken Teilbild sind 10 Seed-Dreiecke ermittelt und gelb dargestellt worden, im mittleren 30 und im rechten 50. Da die Seed-Dreiecke in etwa gleichmäßig über das gesamte dreidimensionale Objekt verteilt werden, ist jeweils nur eine Teilmenge von ihnen sichtbar.

[2] Beim `EvOpSeg-Tool` wird hierfür der Schwerpunkt der Netzknoten verwendet.

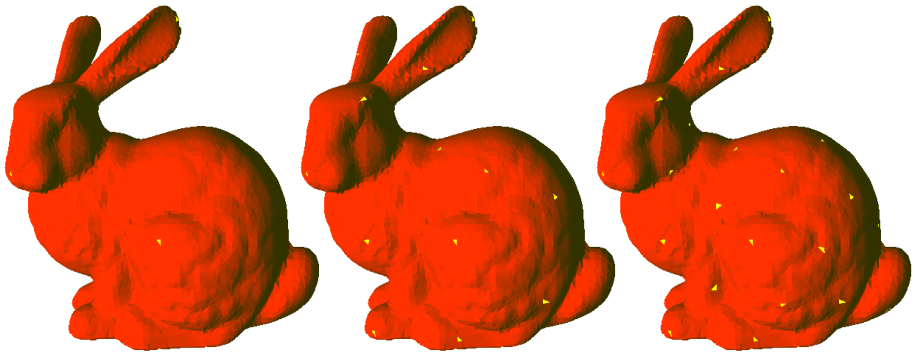


Abbildung 5.2: *Anordnung der Seed-Dreiecke (10, 30 und 50 Seed-Dreiecke).*

5.1.3 Hybrides Initial-Seeding

Beim *hybriden Initial-Seeding* werden einige Seeds manuell und andere automatisch gesetzt. Die Reihenfolge ist dabei prinzipiell irrelevant. Es ist sogar möglich, den Prozess beliebig zu iterieren, also beispielsweise einige Seeds manuell anzugeben, dann weitere automatisch hinzuzufügen und anschließend erneut Seeds manuell zu platzieren. Wenn zu Beginn des automatischen Initial-Seeding-Schritts bereits Seeds vorhanden sind, werden diese direkt berücksichtigt. Das bedeutet, dass auch der erste automatisch zu ermittelnde Seed-Punkt so angeordnet wird, dass er von allen vorhandenen möglichst weit entfernt ist. Auf die Ermittlung der zentral gelegenen Facette t_0 sowie der Facette t_1, die von t_0 den maximalen Abstand hat, wird also verzichtet. Andernfalls wäre nicht auszuschließen, dass zwei Seeds sehr nahe zusammen oder sogar auf derselben Facette liegen.

5.2 Patch-Ermittlung

Der Patch-Ermittlung liegt der Segmentierungsbegriff aus Definition 2.8 zugrunde, nach der die Menge der Mesh-Dreiecke partitioniert wird. Jedem Seed-Punkt werden Mesh-Dreiecke zugeordnet, die insgesamt eine zusammenhängende Oberflächenregion bilden; dabei wird kein Dreieck mehreren Seed-Punkten zugeordnet. Die Zuordnung hat so zu erfolgen, dass jeder Seed-Punkt genau die Dreiecke erhält, die in gewissem Sinne näher an ihm liegen als an allen anderen Seed-Punkten. Dafür wird eine geeignete Distanzfunktion benötigt, welche die Segmentierungskriterien implizit enthält.

In den beiden folgenden Unterabschnitten werden zunächst zwei neuartige Distanzfunktionen definiert, die jeweils für die Patch-Ermittlung geeignet sind. Dabei handelt es sich um die *winkelbasierte Distanz* sowie die *merkmalbasierte Distanz*, wobei letztere eine Erweiterung der ersten darstellt. Für den Algorithmus zur Patch-Berechnung, auf den Abschnitt 5.2.3 ausführlich eingeht, ist es unerheblich, ob die winkelbasierte oder die merkmalsbasierte Distanz verwendet wird.

5.2.1 Winkelbasierte Distanz

Weiter vorne wurde bereits beschrieben, dass Segmentgrenzen erwartet werden, die weitgehend durch konkave Objektregionen verlaufen. Die *winkelbasierte Distanzfunktion* berücksichtigt diese Erwartung, indem die geodätische Distanz auf der Oberfläche des gegebenen Dreiecksnetzes mit einer Distanz kombiniert wird, die sich aus dem Winkel zwischen benachbarten Dreiecken ergibt. Sie ist definiert als

$$d^{\zeta}_{ang}(t_1, t_2) := d_{geo}(t_1, t_2) + \zeta \cdot d_{\measuredangle}(t_1, t_2), \tag{5.2}$$

wobei t_1 und t_2 *benachbarte Dreiecke* sind. Ähnlich wie in [KT03] kombiniert auch diese Funktion die geodätische Distanz d_{geo} mit der *Winkel-Distanz* $d_{\measuredangle}$. Dabei steuert $\zeta \in \mathbb{R}^+_0$ den Einfluss der Winkel-Distanz, die durch

$$d_{\measuredangle}(t_1, t_2) := \max(\alpha(t_1, t_2), 0) \tag{5.3}$$

definiert ist. Mit $\alpha(t_1, t_2)$ wird der vorzeichenbehaftete Winkel zwischen den Normalenvektoren der benachbarten Dreiecke t_1 und t_2 bezeichnet, der mit dem dihedralen Winkel zwischen den Dreiecken zusammenhängt (vgl. auch Abschnitt 2.2); das Vorzeichen gibt dabei an, ob das Objekt an der gemeinsamen Kante konvex ($\alpha \leq 0$) oder konkav ($\alpha > 0$) ist. Die Nichtberücksichtigung des Winkels bei konvexen Kanten unterscheidet diese Distanzfunktion von derjenigen, die Katz und Tal in [KT03] angegeben haben. Auf diese Weise wird beim EvOpSeg-Verfahren versucht, konkaven Oberflächenregionen einen noch größeren Einfluss zu geben.

Die *winkelbasierte Distanz* $d^{\zeta}_{ang}(t, t')$ *zwischen nicht-benachbarten Dreiecken* t und t' ist definiert als die minimale winkelbasierte Länge aller Pfade zwischen diesen Dreiecken. Ein *Pfad*

$$\Psi := (t_1, t_2, \ldots, t_m),\ t_1 = t,\ t_m = t', \tag{5.4}$$

zwischen den Dreiecken t und t' ist definiert als eine Sequenz von Dreiecken, des gegebenen Netzes M, wobei zwei in der Sequenz aufeinander folgende Dreiecke auch in M benachbart sind. Die *winkelbasierte Länge* eines Pfades Ψ, der m Dreiecke enthält, ist gegeben durch

$$l^{\zeta}(\Psi) := \sum_{i=1}^{m-1} d^{\zeta}_{ang}(t_i, t_{i+1}). \tag{5.5}$$

Der Grund für die Kombination der geodätischen Distanz d_{geo} mit der Winkel-Distanz $d_{\measuredangle}$ liegt in der Tatsache, dass geodätische Distanzen Krümmungen unberücksichtigt lassen. Somit würde eine ausschließliche Verwendung des geodätischen Abstandes bei der in Abschnitt 5.2.3 beschriebenen Patch-Berechnung dazu führen, dass die Grenzen der logischen Komponenten nicht oder nicht gut approximiert werden. Eine reine Berücksichtigung der Winkel-Distanz ist ebenfalls nicht zu empfehlen, da sie nicht dafür geeignet ist, Komponentengrenzen zu finden, die durch ebene Objektregionen verlaufen. Bei einer konkaven Kante ist die Winkel-Distanz relativ groß, wodurch üblicherweise auch ein verhältnismäßig großer winkelbasierter Distanzwert resultiert. Das führt dazu, dass bei der eigentlichen Patch-Berechnung konkave Netzkanten tendenziell seltener überquert werden als Kanten, die in konvexen Bereichen liegen.

5.2.2 Merkmalbasierte Distanz

In diesem Unterabschnitt wird eine weitere Distanz definiert, die sich zur Berechnung von Patches eignet. Formal handelt es sich um eine Erweiterung der winkelbasierten Distanz, bei der auch Informationen über die Gestalt des Objekts bzw. seiner Teile berücksichtigt werden. Simari et al. lassen die Anwendenden in [SNKS09] Informationen über Aufbau und Geometrie des Objekts mit einbringen. Dabei wird beispielsweise angegeben, dass eine Komponente vorhanden ist, die flach oder schmal ist. Die vorliegende Dissertation verfolgt einen Ansatz, der ohne solche Angaben auskommt. Dadurch bietet er sich auch für eine vollkommen automatische Segmentierung ohne manuelle Eingriffe an.

Wenn ein Modellteil, also eine Komponente, bereits bekannt ist, kann dessen Form recht einfach automatisch analysiert werden. Dies bedeutet aber, dass im Prinzip bereits eine Segmentierung vorliegt. Da eine solche Segmentierung bei dem EvOpSeg-Ansatz jedoch erst noch zu ermitteln ist, müssen Aussagen über die Formen möglicher Teile anhand lokaler Oberflächeneigenschaften getroffen werden. Offensichtlich sind die Hauptkrümmungen dafür gut geeignet. Bei ihnen handelt es sich um intrinsische Eigenschaften [CS92], die einige Rückschlüsse auf die Form erlauben. Die Bestimmung der Hauptkrümmungen kann über so genannte Normalkrümmungen erfolgen. Unter einer *Normalkrümmung* einer Fläche f an einem Punkt $\mathbf{p}$ der Fläche versteht man die Krümmung der Schnittkurve im Punkt $\mathbf{p}$, die entsteht, wenn f mit der Ebene geschnitten wird, die von dem Normalenvektor der Fläche am Punkt $\mathbf{p}$ und einer Tangente an diesem Punkt aufgespannt wird [AM91, S. 91]. Bei differenzierbaren Flächen existieren unendlich viele solcher Tangenten und somit auch unendlich viele Normalkrümmungen. Zwei Normalkrümmungen kommt eine besondere Bedeutung zu: die Normalkrümmung mit dem maximalen Wert ist die *erste Hauptkrümmung* κ_1, die Normalkrümmung mit dem minimalen Wert ist die *zweite Hauptkrümmung* κ_2. Beide Hauptkrümmungen sind stets orthogonal zueinander [TCG08] und werden auch in dieser Arbeit zur Bestimmung formbasierter Merkmale herangezogen.

Bei Oberflächen, die zweimal stetig differenzierbar sind, stellt die Berechnung der Hauptkrümmungen kein großes Problem dar. Dies trifft insbesondere auf Flächen zu, denen eine analytische Repräsentation zugrunde liegt. Anders sieht es hingegen bei Polygonnetzen aus: Dort müssen die Hauptkrümmungen geeignet abgeschätzt werden. Im Idealfall entsprechen die abgeschätzten Krümmungen den tatsächlichen Krümmungen der durch das Polygonnetz approximierten Objektoberfläche. Diese Problemstellung ist jedoch keineswegs trivial, weswegen immer wieder neue Verfahren für Polygonnetze entwickelt werden [DKT05, KSNS07]. Die meisten bekannten Verfahren ermitteln die Hauptkrümmungen an den Netzknoten. Es gibt allerdings auch andere Ansätze, bei denen nicht den Knoten sondern den Facetten Krümmungswerte zugewiesen werden [SN06]. Chen und Schmitt haben ein Verfahren angegeben, das auf dem Euler-Theorem basiert [CS92]. Das Verfahren bildet die Grundlage für den Algorithmus von Dong und Wang [DW05], der als Ergebnis eine sehr gute Abschätzung der Krümmungswerte an den Knoten liefert und für die Definition der merkmalbasierten Distanz herangezogen wurde.

Zur Herleitung einer merkmalbasierten Distanzfunktion, die für die Segmentierung von Dreiecksnetzen geeignet ist, werden vier Schritte nacheinander durchgeführt:

1. **Krümmungsberechnung:** Für jeden Knoten des gegebenen Dreiecksnetzes werden die beiden Hauptkrümmungen berechnet. In dieser Arbeit ist dafür das von Dong und Wang beschriebene Verfahren [DW05] verwendet worden.
2. **Krümmungsbasierte Form-Klassifikation:** Ausgehend von den im ersten Schritt berechneten Hauptkrümmungen wird für jeden Knoten ermittelt, welche Form die den Knoten umgebende Netzregion hat.
3. **Bestimmung formbasierter Merkmalsvektoren:** Für jedes Dreieck des gegebenen Dreiecksnetzes wird ein so genannter formbasierter Merkmalsvektor bestimmt. Dies geschieht mit Hilfe des im zweiten Schritt erhaltenen Klassifikationsergebnisses.
4. **Definition der merkmalbasierten Distanzfunktion:** Die formbasierten Merkmalsvektoren erlauben die Definition einer merkmalbasierten Distanzfunktion, die bewirkt, dass bei der Patch-Berechnung auch Grenzen zwischen unterschiedlich geformten Objektteilen berücksichtigt werden.

Diese vier Schritte werden im Folgenden ausführlicher erläutert.

Schritt 1: Krümmungsberechnung nach Dong und Wang

Die im Rahmen dieser Dissertation verwendete Berechnung der Hauptkrümmungen basiert auf dem Ansatz von Dong und Wang [DW05]. Im Unterschied zu diesem wird hier durch die Berücksichtigung anderer Nachbarschaften versucht, den Einfluss schlecht gescannter oder verrauschter Netzknoten möglichst gering zu halten. Abgesehen davon und von einer an die Konventionen dieser Arbeit angepassten Notation sind die Formeln (5.6) bis (5.22) mit zugehörigem Inhalt aus [DW05] übernommen. Der Algorithmus von Dong und Wang zur Bestimmung der Hauptkrümmungen besteht im Wesentlichen aus zwei Schritten:

1. Ermittlung geeigneter Normalenvektoren für die Knoten.
2. Ermittlung der Hauptkrümmungen an den Knoten mit Hilfe der im ersten Schritt erhaltenen Normalenvektoren.

Im ersten Schritt wird für jeden Knoten des gegebenen Dreiecksnetzes M ein Normalenvektor $\vec{n}$ ermittelt. Dazu werden die Normalenvektoren aller zu einem solchen Knoten adjazenten Dreiecke geeignet gemittelt. Für einen Knoten v erfolgt die Berechnung des zugehörigen Normalenvektors durch

$$\vec{n} := \frac{\sum_{i=1}^{m} w_i \cdot \vec{n}_{t_i}}{|| \sum_{i=1}^{m} w_i \cdot \vec{n}_{t_i} ||}, \tag{5.6}$$

wobei m die Anzahl der zu v adjazenten Dreiecke angibt und mit t_i das i-te solche Dreieck sowie mit $\vec{n}_{t_i}$ der Einheitsnormalenvektor von t_i bezeichnet wird. Dabei wird davon ausgegangen, dass alle solchen Einheitsnormalenvektoren zur selben Seite der Oberfläche – bei geschlossenen Oberflächen also beispielsweise alle nach „außen“ – zeigen. Die Gewichte w_i sind definiert als

$$w_i := \frac{1}{||\vec{g}_i - \vec{p}||}, \ (i = 1, \ldots, m), \tag{5.7}$$

wobei $\vec{g}_i$ den Stützvektor zum Schwerpunkt $\mathbf{g}_i$ des Dreiecks t_i und $\mathbf{p}$ den zum Knoten v gehörenden Punkt sowie $\vec{p}$ dessen Stützvektor bezeichnen.

Nachdem für jeden Knoten ein Normalenvektor bestimmt worden ist, lassen sich mit diesen Informationen nun im zweiten Schritt die beiden Hauptkrümmungen κ_1 und κ_2 ermitteln. Dazu projizieren Dong und Wang für jeden direkten Nachbarn $\mathbf{p}_i$ des zum aktuell betrachteten Knoten gehörenden Punktes $\mathbf{p}$ den Vektor $\vec{p}_i - \vec{p}$, der sich aus der Differenz der beiden Stützvektoren ergibt, auf die Tangentialebene in $\mathbf{p}$ – also auf die Ebene, die orthogonal zum Normalenvektor $\vec{n}$ von $\mathbf{p}$ verläuft und die $\mathbf{p}$ beinhaltet – und normieren anschließend den durch die Projektion entstandenen Vektor. Der sich so ergebende normierte Vektor ist durch

$$\vec{t}_i := \frac{(\vec{p}_i - \vec{p}) - \langle \vec{p}_i - \vec{p}, \vec{n} \rangle \cdot \vec{n}}{||(\vec{p}_i - \vec{p}) - \langle \vec{p}_i - \vec{p}, \vec{n} \rangle \cdot \vec{n}||} \tag{5.8}$$

gegeben. Anstelle der von Dong und Wang verwendeten direkten Nachbarn, also der 1-Ring-Nachbarschaft, können auch größere Nachbarschaften genutzt werden. Größere Nachbarschaften haben den Vorteil, dass der Einfluss von Störungen bzw. Rauschen verringert wird [SN06, Rus04]. Dies kann bei einigen Modellen, insbesondere wenn sie durch eine nicht optimale Abtastung der Objekte entstanden sind, zu besseren Ergebnissen führen. Aus diesem Grund ist auch in der vorliegenden Dissertation von der 1-Ring-Nachbarschaft abgesehen worden: Stattdessen werden alle Knoten $\mathbf{p}_i$ (jeweils mit Stützvektor $\vec{p}_i$ und Einheitsnormalenvektor $\vec{n}_i$) betrachtet, die von dem aktuellen den Abstand 3 haben.

Für jedes $\vec{t}_i$ eines Punktes $\mathbf{p}$ wird die Normalkrümmung $\kappa_n(\vec{t}_i)$ berechnet:

$$\kappa_n(\vec{t}_i) = -\frac{\langle \vec{p}_i - \vec{p}, \vec{n}_i - \vec{n} \rangle}{\langle \vec{p}_i - \vec{p}, \vec{p}_i - \vec{p} \rangle}, \ (i = 1, \ldots, m). \tag{5.9}$$

Der Vektor $\vec{t}_i$, für den der Funktionswert am größten ist, wird im Folgenden mit $\vec{t}_{id}$ bezeichnet. In der durch $\mathbf{p}$ verlaufenden Tangentialebene wird ein spezielles Koordinatensystem gewählt, dessen Achsen durch

$$\vec{\hat{e}}_1 := \vec{t}_{id} \text{ und } \vec{\hat{e}}_2 := \frac{\vec{\hat{e}}_1 \times \vec{n}}{||\vec{\hat{e}}_1 \times \vec{n}||} \tag{5.10}$$

gegeben sind. Es sei θ_i der Winkel zwischen $\vec{t}_i$ und $\vec{\hat{e}}_1$. Damit kann dann $\kappa_n(\vec{t}_i)$ für $i \in \{1, \ldots, m\}$ auch anders angegeben werden:

$$\kappa_n(\vec{t}_i) = a\cos^2(\theta_i) + b\cos(\theta_i)\sin(\theta_i) + c\sin^2(\theta_i). \tag{5.11}$$

Dong und Wang geben an, wie die Werte von a, b und c zu bestimmen sind, mit denen schließlich die Hauptkrümmungen berechnet werden können:

$$a := \kappa_n(\vec{t}_{id}) \tag{5.12}$$

$$b := \frac{a_{13}a_{22} - a_{23}a_{12}}{a_{11}a_{22} - (a_{12})^2} \tag{5.13}$$

$$c := \frac{a_{11}a_{23} - a_{12}a_{13}}{a_{11}a_{22} - (a_{12})^2}. \tag{5.14}$$

Die darin vorkommenden Parameter sind wie folgt zu wählen:

$$a_{11} := \sum_{i=1}^{m} \cos^2(\theta_i)\sin^2(\theta_i) \tag{5.15}$$

$$a_{12} := \sum_{i=1}^{m} \cos(\theta_i)\sin^3(\theta_i) \tag{5.16}$$

$$a_{13} := \sum_{i=1}^{m} (\kappa_n(\vec{t}_i) - a\cos^2(\theta_i))\cos(\theta_i)\sin(\theta_i) \tag{5.17}$$

$$a_{21} := a_{12} \tag{5.18}$$

$$a_{22} := \sum_{i=1}^{m} \sin^4(\theta_i) \tag{5.19}$$

$$a_{23} := \sum_{i=1}^{m} (\kappa_n(\vec{t}_i) - a\cos^2(\theta_i))\sin^2(\theta_i). \tag{5.20}$$

Mit diesen Werten berechnen Dong und Wang dann die beiden Hauptkrümmungen nach

$$\kappa_1 = H + \sqrt{H^2 - K_G},\ \kappa_2 = H - \sqrt{H^2 - K_G}, \tag{5.21}$$

wobei H und K_G als Approximationen der mittleren Krümmung und der Gaußschen Krümmung gegeben sind durch:

$$H := \frac{a+c}{2},\ K_G := \frac{ac - b^2}{4}. \tag{5.22}$$

Auch die Richtungen der Hauptkrümmungen lassen sich recht einfach bestimmen. Diese spielen jedoch für die krümmungsbasierte Distanz keine weitere Rolle, so dass hier auch nicht näher darauf eingegangen wird.

Schritt 2: Krümmungsbasierte Form-Klassifikation

Nachdem im ersten Schritt die beiden Hauptkrümmungen an den Knoten des gegebenen Dreiecksnetzes approximiert worden sind, kann nun für jedes Element des Dreiecksnetzes – insbesondere für jeden Knoten und jede Fläche – eine Form-Klassifikation durchgeführt werden. Dabei wird anhand der Hauptkrümmungen abgeschätzt, welche Gestalt

das Objekt in der lokalen Umgebung der betrachteten Stelle hat. Die Klassifikations-Ergebnisse werden später zur Definition von formbasierten Merkmalsvektoren genutzt.

Zu jedem Knoten des Netzes sind die beiden Hauptkrümmungen κ_1 und κ_2 wie oben beschrieben ermittelt worden. Im Folgenden wird mit κ_1^i die erste Hauptkrümmung des i-ten Knoten des Dreiecksnetzes und mit κ_2^i die zweite Hauptkrümmung dieses Knotens bezeichnet. Da die Hauptkrümmungen nahezu beliebige Werte annehmen können, erfolgt zunächst eine Normierung auf den Wertebereich $[0, 1]$. Vergne et al. nutzen als Normierungsfunktion den Tangens Hyperbolicus [VBGS08]. Experimente haben jedoch gezeigt, dass sich eine Normierung mit Hilfe des Tangens Hyperbolicus nicht besonders gut für die Definition der gewichteten merkmalbasierten Distanz eignet, da relativ viele Krümmungswerte auf Werte abgebildet werden, die sehr nah an 1 bzw. -1 liegen. Als wesentlich besser geeignet hat sich eine lineare Skalierung erwiesen, bei der

$$\kappa_{max} := \max\{|\kappa_j^i| \mid i \in \{1, \ldots, |V|\},\ j \in \{1, 2\}\} \tag{5.23}$$

den Absolutwert der betragsmäßig größten aller für das Modell ermittelten Hauptkrümmungen angibt, wobei V die Menge der Knoten des Dreiecksnetzes sei. Durch

$$\kappa_{j,norm}^i := \frac{\kappa_j^i}{\kappa_{max}},\ i \in \{1, \ldots, |V|\},\ j \in \{1, 2\} \tag{5.24}$$

wird κ_j^i linear auf einen Wert im Intervall $[-1, 1]$ abgebildet.

Bisher sind Krümmungswerte lediglich für die Knoten des Dreiecksnetzes definiert. Um auch für die Dreiecke approximierte Hauptkrümmungen angeben zu können, wird jedem Dreieck das arithmetische Mittel der durch (5.24) normierten κ_1-Werte seiner Eckpunkte als erste normierte Hauptkrümmung des Dreiecks sowie das arithmetische Mittel der entsprechenden κ_2-Werte als zweite normierte Hauptkrümmung zugewiesen. Diese zugewiesenen Werte werden im Folgenden als die *(normierten) Hauptkrümmungen* κ_1 und κ_2 des Dreiecks angesehen.

Anhand der normierten Hauptkrümmungen kann sowohl für Netzknoten als auch für Dreiecke eine Form-Klassifikation erfolgen. Dazu werden für ein entsprechendes Netzelement (Knoten oder Facette) dessen normierte Hauptkrümmungen κ_1 und κ_2 betrachtet. Sind beide betragsmäßig relativ klein, liegt das Netzelement in einer ebenen Region. Dies ist in Abbildung 5.3 als gelber Halbkreis dargestellt. Da nach Konstruktion stets $\kappa_1 \geq \kappa_2$ gilt, braucht der Bereich oberhalb der Geraden $\kappa_2 = \kappa_1$ im Gegensatz zu einer ähnlichen Darstellung in [VBGS08] nicht weiter berücksichtigt zu werden. Krümmungswerte, die auf konkave Stellen schließen lassen, sind im roten Bereich zu finden. Ein Netzelement, für das $(\kappa_1, \kappa_2)^T$ in diesen Bereich fällt, wird entsprechend als konkav klassifiziert. Durch Spiegelung des roten Bereichs an der Geraden $\kappa_2 = -\kappa_1$ erhält man die Krümmungswerte, die auf konvexe (grün) oder zylinderförmige (lila) Stellen hinweisen. Wenn κ_1 und κ_2 betragsmäßig ungefähr gleich groß sind, aber unterschiedliche Vorzeichen haben (blauer Bereich in der Abbildung), deutet dies auf einen Sattelpunkt hin. Weitere Unterscheidungen, beispielsweise ob konkave oder konvexe Teile in beide Hauptrichtungen ähnliche Krümmungen aufweisen und es sich somit wie in [VBGS08] angegeben um innen bzw. außen liegende Regionen von zumindest

abschnittsweise kugelähnlich gewölbten Komponenten handelt, scheinen für die Segmentierung dreidimensionaler Modelle nicht relevant zu sein.

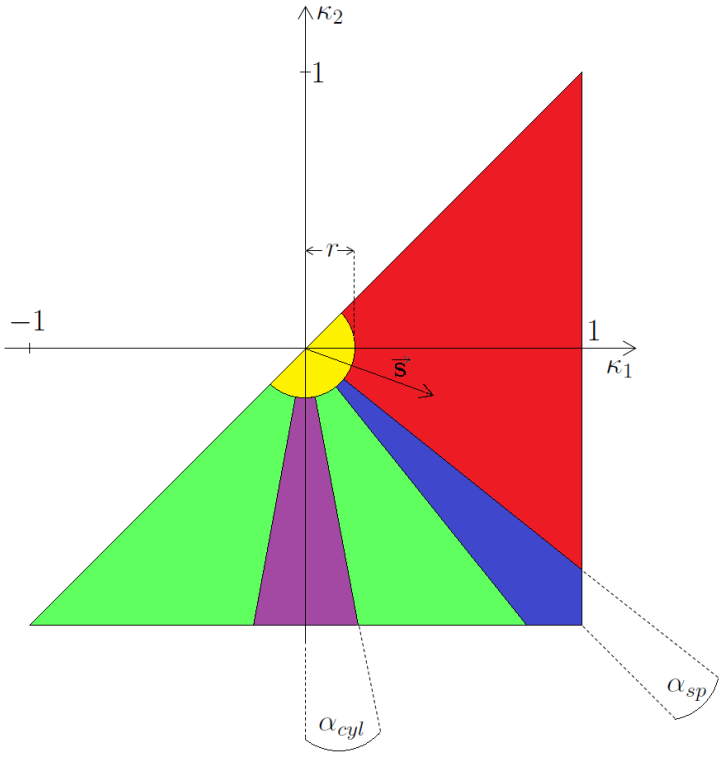


Abbildung 5.3: *Zusammenhang zwischen den normierten Hauptkrümmungen von Facetten des Dreiecksnetzes und der daraus resultierenden Form-Schätzung von Objektteilen. Unterschieden werden ebene Regionen (gelb), Sattelpunkte (blau) sowie konvexe (grün), zylinderförmige (lila) und konkave Bereiche (rot).*

Es sei $\vec{s}$ ein Vektor, dessen Koordinaten durch die normierten Hauptkrümmungen des betrachteten Netzelements gegeben sind. Dieser Vektor wird im Folgen als *Krümmungsbeschreibungsvektor* bezeichnet. Der blaue, für Sattelpunkte stehende Bereich aus Abbildung 5.3 ist durch alle Koordinaten gegeben, für die $|\vec{s}| \geq r$ ist und für die zudem gilt, dass $\vec{s}$ mit dem Vektor $(1, -1)^T$ einen Winkel einschließt, der kleiner als α_{sp} ist. Analog dazu gilt für den lila gefärbten Bereich, dass $|\vec{s}| \geq r$ ist und $\vec{s}$ mit dem Vektor $(0, -1)^T$ einen Winkel einschließt, der kleiner als α_{cyl} ist.

In Abbildung 5.4 sind zwei Ergebnisse von bezüglich der Dreiecke vorgenommenen Form-Klassifikationen zu sehen. Die Interpretation der Farben erfolgt analog zu Abbildung 5.3: Gelbe Bereiche wurden als eben klassifiziert, rote als konkav, grüne als konvex, lila als zylinderförmig und blaue als Bereiche mit Sattelpunkten. Die Parameter sind als $r = 0.01$, $\alpha_{cyl} = 16.65°$ und $\alpha_{sp} = 19.2°$ gewählt worden, was bei zahlreichen Modellen zu guten Ergebnissen führt. Die Tischbeine werden damit im Wesentlichen als zylinderförmig klassifiziert, ihre Fußenden als konvex. Ähnliches gilt auch für die

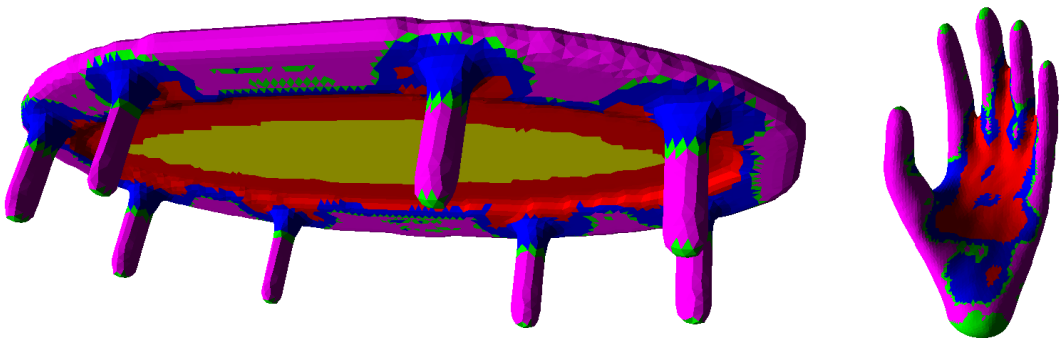


Abbildung 5.4: *Ergebnisse der krümmungsbasierten Form-Klassifikation.*

Finger sowie deren Kuppen der im rechten Teilbild dargestellten Hand. Der mittlere Bereich der Tischplattenunterseite ist etwas nach innen gewölbt. Diese konkave Wölbung bewirkt die Rotfärbung der um den entsprechenden Bereich herumliegenden Dreiecke. Aus Übergängen von zylinderförmigen zu ebenen Teilen resultieren im Allgemeinen Sattelpunkte, wie auch deutlich an den Verbindungsstellen zwischen den Tischbeinen und der Tischplatte zu erkennen ist.

Schritt 3: Formbasierte Merkmalsvektoren

Mit Hilfe der beschriebenen Klassifikationen können Formmerkmale definiert und in so genannten formbasierten Merkmalsvektoren zusammengefasst werden. Diese formbasierten Merkmalsvektoren ermöglichen die Definition einer merkmalbasierten Distanzfunktion, die für zwei Dreiecke einen verhältnismäßig kleinen Distanzwert liefern soll, wenn ein Pfad zwischen beiden existiert, der ausschließlich Dreiecke mit gleichen oder zumindest ähnlichen Formmerkmalen enthält. Gibt es keinen solchen Pfad, kann das ein Indiz dafür sein, dass beide Dreiecke zu unterschiedlichen Komponenten gehören; der Distanzwert sollte dann entsprechend größer ausfallen.

Bei der oben beschriebenen Ermittlung der Hauptkrümmungen von Dreiecken können Klassifikationsinformationen der Netzknoten durch Berechnung des arithmetischen Mittels verloren gehen. Angenommen, zwei der drei Eckpunkte eines Dreiecks t sind als in einer Ebene liegend klassifiziert worden und für den Krümmungsbeschreibungsvektor $\vec{s}$ des dritten gelte $|\vec{s}| > 5 \cdot r$. Dann würde die Länge des sich für t ergebenden Krümmungsbeschreibungsvektors in jedem Fall echt größer als r sein und t somit als nicht in einer Ebene liegend klassifiziert werden, obwohl die Klassifikation bezüglich der Netzknoten dies für zwei seiner drei Eckpunkte ermittelt hat. Insbesondere für kleine Radien r kann dies sehr schnell vorkommen. Aus diesem Grund werden zunächst formbasierte Merkmalsvektoren für die Knoten bestimmt, und erst anschließend aus diesen die formbasierten Merkmalsvektoren der Dreiecke abgeleitet. Auf diese Weise werden sämtliche Klassifikationsinformationen berücksichtigt.

Für jeden Knoten des Netzes seien die normierten Hauptkrümmungen κ_1 und κ_2 sowie die daraus hergeleitete krümmungsbasierte Form-Klassifikation bekannt. Während der

Wertebereich der Hauptkrümmungen unendlich viele Elemente umfasst, kommen für die krümmungsbasierte Form-Klassifikation nur fünf Werte in Frage: konkav, konvex, eben, zylinderförmig oder in der Nähe eines Sattelpunktes liegend. Anhand dieser Werte wird zunächst für jeden Knoten v ein formbasierter Merkmalsvektor erzeugt.

Zur Bestimmung des formbasierten Merkmalsvektors eines Knotens v wird die Menge $V_{neighbor}(v)$ aller Knoten bestimmt, die von v aus über maximal d_{max} viele Kanten des Dreiecksnetzes erreichbar sind; v ist dabei ebenfalls ein Element der Menge $V_{neighbor}$. Es sei z_{convex} die Anzahl der Knoten aus $V_{neighbor}$, die als konvex klassifiziert worden sind. Analog seien $z_{concave}$, $z_{cylinder}$, z_{plane} und z_{saddle} die Anzahlen der als konkav, zylinderförmig, eben und zu einer Sattelpunktregion gehörend klassifizierten Knoten. Dann ist der *formbasierte Merkmalsvektor* definiert als

$$\vec{f}_{shape}(v) := \frac{1}{|V_{neighbor}(v)|} \cdot \begin{pmatrix} z_{convex} \\ z_{concave} \\ z_{cylinder} \\ z_{plane} \\ z_{saddle} \end{pmatrix}. \tag{5.25}$$

Es seien $\vec{f}_{shape}(v_1)$, $\vec{f}_{shape}(v_2)$ und $\vec{f}_{shape}(v_3)$ die formbasierten Merkmalsvektoren der zu den drei Eckpunkten eines Dreiecks t gehörenden Knoten v_1, v_2 und v_3. Dann ist

$$\vec{f}_{shape}(t) := \frac{\vec{f}_{shape}(v_1) + \vec{f}_{shape}(v_2) + \vec{f}_{shape}(v_3)}{3} \tag{5.26}$$

der *formbasierte Merkmalsvektor* des Dreiecks t.

Schritt 4: Definition der merkmalbasierten Distanzfunktion

Die formbasierten Merkmalsvektoren der einzelnen Dreiecke ermöglichen die Definition einer merkmalbasierten Distanzfunktion, die dafür sorgt, dass Dreieckszügen ein größerer Distanzwert zugewiesen wird, wenn über unterschiedlich geformte Objektteile gelaufen wird. Analog zur winkelbasierten Distanz liegt der merkmalbasierten Distanzfunktion eine so genannten *merkmalbasierten Länge eines Dreieckszugs* zugrunde. Bei der winkelbasierten Länge eines Dreieckszugs wurde zwar der Winkel zwischen benachbarten Dreiecken des Dreieckszugs berücksichtigt, nicht jedoch die Objektform in der Umgebung. Dies ist bei der merkmalbasierten Länge anders.

Es sei Ψ wie in Formel (5.4) ein Dreieckszug von t_1 nach t', der m Elemente enthält. Die *merkmalbasierte Länge von* Ψ ist definiert als

$$l^{\zeta,\eta}_{feature}(\Psi) := \sum_{i=1}^{m-1} d^{\zeta,\eta}_{feature}(t_i, t_{i+1}), \tag{5.27}$$

wobei $d^{\zeta,\eta}_{feature}(t_i, t_{i+1})$ die *merkmalbasierte Distanz* zweier benachbarter Dreiecke angibt. Die *merkmalbasierte Distanzfunktion* $d^{\zeta,\eta}_{feature}$ ist für zwei benachbarte Dreiecke t_i und t_j als Erweiterung der winkelbasierten Distanzfunktion wie folgt definiert:

$$d^{\zeta,\eta}_{feature}(t_i, t_j) := \underbrace{d_{geo}(t_i, t_j) + \zeta \cdot d_{\measuredangle}(t_i, t_j)}_{d^{\zeta}_{ang}(t_i, t_j)} + \eta \cdot d_{shape}(t_i, t_j);\ \zeta, \eta \in \mathbb{R}^+_0. \tag{5.28}$$

Die additive Verknüpfung der ersten beiden Summanden entspricht der winkelbasierten Distanzfunktion aus Gleichung (5.2). Mit ζ und η wird gesteuert, welchen Einfluss $d_{\measuredangle}$ bzw. d_{shape} haben. Die in Gleichung (5.28) verwendete Funktion d_{shape} gibt an, wie stark sich die formbasierten Merkmalsvektoren der beiden benachbarten Dreiecke unterscheiden. Wenn t_i und t_j zu derselben Komponente gehören und nicht allzu nahe am Komponentenrand liegen, werden sich die formbasierten Merkmalsvektoren nicht großartig unterscheiden. Anders sieht es hingegen aus, wenn beide Dreiecke am Rand der Komponente liegen oder sogar zwei unterschiedlichen Komponenten angehören. Motiviert durch diese Überlegung wird d_{shape} für zwei benachbarte Dreiecke t_i und t_j wie folgt definiert:

$$d_{shape}(t_i, t_j) := \|\vec{f}_{shape}(t_i) - \vec{f}_{shape}(t_j)\|. \tag{5.29}$$

Nach Konstruktion enthält der formbasierte Merkmalsvektor eines Knotens v ausschließlich Werte aus dem Intervall $[0, 1]$. Dasselbe gilt nach Gleichung (5.26) auch für den formbasierten Merkmalsvektor eines Dreiecks t. Da darüber hinaus stets $\|\vec{f}_{shape}(t)\| \leq 1$ ist, lässt sich leicht zeigen, dass der Wertebereich von d_{shape} durch das Intervall $[0, \sqrt{2}]$ gegeben ist.

5.2.3 Algorithmus

Anschaulich formuliert werden bei der Patch-Berechnung jedem Seed-Punkt alle Dreiecke des Polygonnetzes zugeordnet, für die er der nächstgelegene Seed-Punkt ist. Bei einem Seed-Punkt, der keinem Seed-Cluster angehört, bildet die Menge der so zugeordneten Dreiecke ein *Patch*. Das Patch eines Seed-Clusters ist hingegen gegeben durch die Menge aller Dreiecke, die einem Seed des Clusters zugeordnet sind. Als Distanzfunktion wird eine gewichtete Variante der winkelbasierten bzw. der merkmalbasierten Distanz verwendet. Ähnlich wie bei gewichteten Voronoi-Diagrammen [AE84, SNKS09] erhält jeder Seed-Punkt ein Gewicht; je größer der Gewichtswert ist, umso größer ist der Einfluss des Seeds und damit auch das von ihm definierte Patch bzw. im Falle von Seed-Clustern der entsprechende Patch-Ausschnitt. Sei $\gamma_{seed} \in \mathbb{R}^+$ das Gewicht eines Seed-Punktes $\mathbf{p}_{seed}$, und t_{seed} das korrespondierende Seed-Dreieck. Dann ist die *gewichtete winkelbasierte Distanz* eines Dreiecks t' von t_{seed} definiert als

$$d_{ang,\gamma}(t', t_{seed}, \gamma_{seed}) := \frac{1}{\gamma_{seed}} \cdot d^{\zeta}_{ang}(t', t_{seed}), \tag{5.30}$$

und die *gewichtete merkmalbasierte Distanz* von t' zu t_{seed} analog als

$$d_{feature,\gamma}(t', t_{seed}, \gamma_{seed}) := \frac{1}{\gamma_{seed}} \cdot d^{\zeta,\eta}_{feature}(t', t_{seed}). \tag{5.31}$$

Welche der beiden Funktionen gewählt wird, ist für den Algorithmus zur Patch-Berechnung vom Prinzip her irrelevant. Deswegen wird im Folgenden vereinfachend stets von einer *gewichteten Distanz* $d_\gamma \in \{d_{ang,\gamma}, d_{feature,\gamma}\}$ gesprochen.

Als erstes wird der Fall betrachtet, dass keine Seed-Cluster vorliegen und jedes Patch somit von genau einem Seed-Punkt definiert wird. Anschließend wird auf Seite 62 auf die Patch-Berechnung bei Seed-Clustern eingegangen.

5.2.3.1 Patch-Berechnung ohne Seed-Cluster

Wenn alle Seeds den gleichen Gewichtswert haben, kann eine Patch-Berechnung erfolgen, indem für jedes Dreieck t_i der Abstand bezüglich des gewählten Abstandmaßes d_γ zu jedem Seed-Dreieck berechnet und t_i dem Patch zugeordnet wird, welches durch das t_i am nächsten liegenden Seed-Dreieck definiert wird. Auf diese Weise ergibt sich automatisch eine Zerlegung des Dreiecksnetzes in disjunkte, zusammenhängende Patches. Bei Verwendung unterschiedlicher Gewichtswerte kann jedoch auf diese Weise nicht mehr gewährleistet werden, dass jedes Patch zusammenhängend ist. Stattdessen können Patches entstehen, die in mehrere nicht-zusammenhängende Teile zerfallen. In Abbildung 5.5 ist ein entsprechendes fehlerhaftes Ergebnis zu sehen. Dabei sind zwei Seed-Punkte gewählt worden; der erste definiert das rote Patch, der zweite das grüne. Im vorliegenden Fall besitzt der erste Seed-Punkt einen doppelt so großen Gewichtswert wie der zweite. Das rote Patch zerfällt in zwei Teile und ist somit auch kein gültiges Patch. Dem Ergebnis lag die gewichtete winkelbasierte Distanz zugrunde.

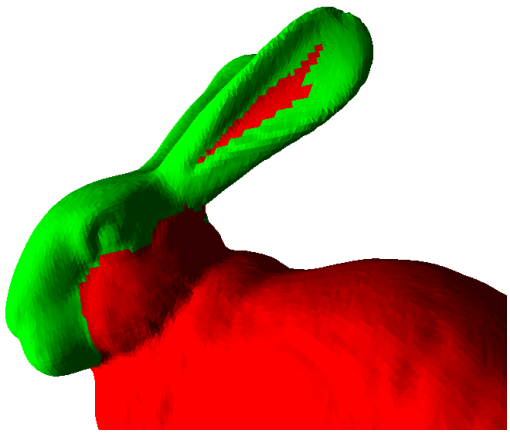


Abbildung 5.5: *Auswirkung einer falschen Berechnungsreihenfolge: Das rote Patch zerfällt in zwei nicht-zusammenhängende Teile.*

Das geschilderte Problem hängt nicht primär mit der Repräsentation des Modells durch Dreiecksnetze und den daraus resultierenden „diskreten“ Abstandsdefinitionen zusammen. Auch im kontinuierlichen Fall ist ein solcher Effekt zu beobachten. Zur Veranschaulichung betrachten wir zunächst den eindimensionalen Raum, der durch die x-Achse des zweidimensionalen Koordinatensystems in Abbildung 5.6 gegeben ist. Es sei

$f(x) := \frac{1}{\gamma_1} \cdot |x - 5|$ eine Funktion, die für das Gewicht $\gamma_1 = 1.0$ die gewichtete Distanz des Punktes $(x, 0)^T$ vom Punkt $\mathbf{p}_1 = (5, 0)^T$ angibt. Ferner sei $g(x) := \frac{1}{\gamma_2} \cdot |x - 8|$ eine entsprechende Funktion, die für das Gewicht $\gamma_2 = 2.0$ einen gewichteten Abstand vom Punkt $\mathbf{p}_2 = (8, 0)^T$ angibt. Die zugehörigen Funktionsgraphen sind in Abbildung 5.6 dargestellt. Es ist zu erkennen, dass für alle Punkte $(x', 0)^T$ mit $2 < x' < 6$ der Abstand zum Punkt $\mathbf{p}_1$ kleiner ist als der zum Punkt $\mathbf{p}_2$, während für alle Punkte $(x'', 0)^T$ mit $x'' < 2$ oder $x'' > 6$ das Gegenteil der Fall ist. Somit ist der Bereich des betrachteten eindimensionalen Raumes, der näher an $\mathbf{p}_2$ als an $\mathbf{p}_1$ liegt, nicht zusammenhängend.

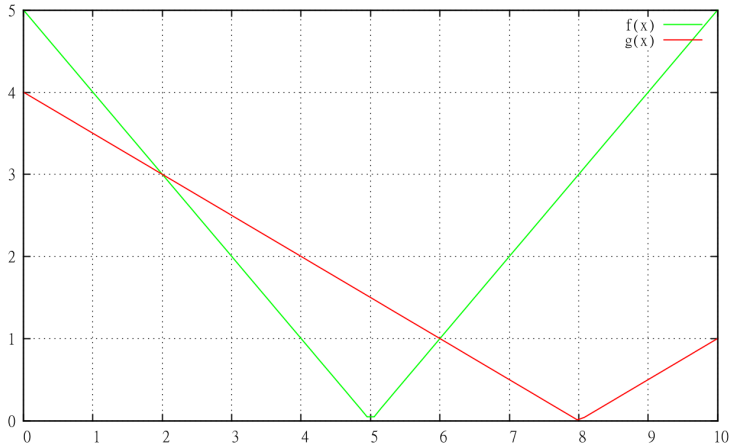


Abbildung 5.6: *Unterschiedlich gewichtete Abstandsfunktionen können zu nicht-zusammenhängenden Bereichen führen.*

Im Gegensatz zum Beispiel aus Abbildung 5.5 wurde bei dem betrachteten eindimensionalen Raum eine stetige Distanzfunktion verwendet. Das beschriebene Ergebnis des eindimensionalen Beispiels lässt sich in ähnlicher Weise auch im zweidimensionalen Raum finden (vgl. bspw. [AE84]) und auf dreidimensionale Modelle mit „glatten“ Oberflächen übertragen. Ein solches dreidimensionales Modell ist beispielsweise ein idealer Zylinder, d.h. ein Zylinder, der analytisch gegeben und nicht etwa durch ein Dreiecksnetz approximiert ist. Auf dem Zylinder können zwei Punkte $\mathbf{p}_1$ und $\mathbf{p}_2$ so angeordnet werden, dass für zwei geeignet gewichtete geodätische Distanzfunktionen $d_{\mathbf{p}_1}$ und $d_{\mathbf{p}_2}$ die Menge aller Oberflächenpunkte des Zylinders, die bzgl. $d_{\mathbf{p}_2}$ näher an $\mathbf{p}_2$ als bzgl. $d_{\mathbf{p}_1}$ an $\mathbf{p}_1$ liegen, zwei nicht-zusammenhängende Bereiche bildet. Der Einfachheit halber werden diese Bereiche im Folgenden ebenfalls als Patches bezeichnet. Für einen Punkt $\mathbf{a}$ auf der Oberfläche des Zylinders werden die gewichteten Abstände zu $\mathbf{p}_1$ bzw. zu $\mathbf{p}_2$ nach

$$d_{\mathbf{p}_1}(\mathbf{a}) := \frac{1}{\gamma_1} \cdot d_{geo}(\mathbf{p}_1, \mathbf{a}), \ \gamma_1 \in \mathbb{R}^+, \tag{5.32}$$

$$d_{\mathbf{p}_2}(\mathbf{a}) := \frac{1}{\gamma_2} \cdot d_{geo}(\mathbf{p}_2, \mathbf{a}), \ \gamma_2 \in \mathbb{R}^+ \tag{5.33}$$

berechnet. Dabei gibt d_{geo} den geodätischen Abstand der jeweiligen Punkte an.

Im Folgenden wird davon ausgegangen, dass $\gamma_2 > \gamma_1$ ist und zudem die Punkte $\mathbf{p}_1$ und $\mathbf{p}_2$, wie im linken Teilbild der Abbildung 5.7 zu sehen ist, untereinander angeordnet

sind. Darüber hinaus wird davon ausgegangen, dass die beiden Punkte den geodätischen Abstand 1 voneinander haben und dass der Umfang der Mantelfläche den Wert 2 hat. Dieser Zusammenhang ist in dem mittleren Teilbild dargestellt. Die abgerollte Mantelfläche ist dort als graue Fläche eingezeichnet.

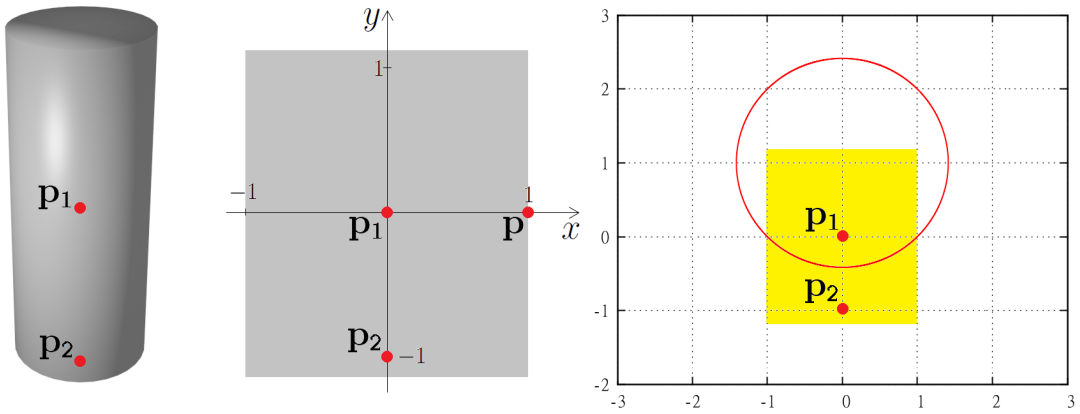


Abbildung 5.7: *Unterschiedliche Gewichte können auch bei dreidimensionalen Modellen mit einer kontinuierlichen Oberfläche zu nicht-zusammenhängenden Bereichen führen.*

Die Gewichte γ_1 und γ_2 sollen nun so gewählt werden, dass die Grenze zwischen den von $\mathbf{p}_1$ und $\mathbf{p}_2$ definierten Patches entlang der Mantelfläche um den Zylinder herum verläuft. Damit erstreckt sich auch das von $\mathbf{p}_1$ definierte Patch um die Mantelfläche herum. Dies ist z.B. dann der Fall, wenn für $\mathbf{p} = (1, 0)^T$ die Ungleichung $d_{\mathbf{p}_1}(\mathbf{p}) < d_{\mathbf{p}_2}(\mathbf{p})$ gilt. Durch Umformen der Ungleichung ergibt sich schließlich, dass $\gamma_2 < \sqrt{2} \cdot \gamma_1$ zu wählen ist.

Zur Grenze zwischen den beiden Patches gehört jeder Punkt $\mathbf{a} = (x, y)^T$ der Oberfläche des Modells, für den gilt:

$$d_{\mathbf{p}_1}(\mathbf{a}) = d_{\mathbf{p}_2}(\mathbf{a}). \tag{5.34}$$

Mit den gegebenen Koordinaten $\mathbf{p}_1 = (0, 0)^T$ und $\mathbf{p}_2 = (0, -1)^T$ führt dies zu:

$$y^2 + \frac{2}{\gamma_2^2 \cdot (\frac{1}{\gamma_2^2} - \frac{1}{\gamma_1^2})} \cdot y + \frac{1}{\gamma_2^2 \cdot (\frac{1}{\gamma_2^2} - \frac{1}{\gamma_1^2})} + x^2 = 0 \tag{5.35}$$

Dabei handelt es sich um eine Kreisgleichung. Mittelpunkt $\mathbf{m}_{circ}$ und Radius r_{circ} ergeben sich wie folgt (vgl. [BSMM99, S. 193]):

$$\mathbf{m}_{circ} = \begin{pmatrix} 0 \\ -\frac{1}{\gamma_2^2 \cdot (\frac{1}{\gamma_2^2} - \frac{1}{\gamma_1^2})} \end{pmatrix}, \; r_{circ} = \sqrt{\left(\frac{1}{\gamma_2^2 \cdot (\frac{1}{\gamma_2^2} - \frac{1}{\gamma_1^2})} \right)^2 - \frac{1}{\gamma_2^2 \cdot (\frac{1}{\gamma_2^2} - \frac{1}{\gamma_1^2})}} \tag{5.36}$$

Für $\gamma_2 = \sqrt{2} \cdot \gamma_1$ ist dieser Kreis im rechten Teilbild der Abbildung 5.7 eingezeichnet. Die Mantelfläche des Zylinders ist gelb angedeutet. Ein in positive Richtung der y-Achse doppelt so hohen Zylinder schneidet auch die oberer Kreishälfte. Dies ist gleichbedeutend damit, dass sich der Abschnitt der Mantelfläche, der oberhalb des Kreises liegt,

gemäß der gewählten Distanzmaße näher an $\mathbf{p}_2$ als an $\mathbf{p}_1$ befindet. Da das von $\mathbf{p}_1$ definierte Patch jedoch um die Mantelfläche herum reicht, zerfällt das zu $\mathbf{p}_2$ gehörende Patch in zwei nicht-zusammenhängende Teile.

Bei den beiden genannten Beispielen aus den Abbildungen 5.6 und 5.7 ist weder ein winkelbasierter noch ein merkmalbasierter Anteil berücksichtigt worden. Trotzdem konnten allein durch die Verwendung unterschiedlicher Gewichte Situationen konstruiert werden, in denen nicht-zusammenhängende Bereiche entstehen. Eine zusätzliche Berücksichtigung eines winkelbasierten oder merkmalbasierten Anteils kann den Effekt noch weiter verstärken. Im Rahmen der Zerlegung eines dreidimensionalen Modells in Patches muss also gesondert sichergestellt werden, dass jedes Patch zusammenhängend ist. Dies kann mit einer speziell an die Problemstellung angepassten Variante des Algorithmus von Dijkstra zur Bestimmung kürzester Wege [Dij59] geschehen, bei der der Dualgraph des Dreiecksnetzes betrachtet wird. Jene Dualgraph-Knoten, die Seed-Dreiecken entsprechen, werden *Seed-Knoten* genannt. Die winkelbasierte bzw. merkmalbasierte Entfernung benachbarter Dreiecke wird im Dualgraph als *Kosten* der entsprechenden Kante notiert. Die Entfernung eines Dualgraph-Knotens a von einem anderen Knoten b ist durch die Kosten eines günstigsten Weges von a nach b gegeben und identisch mit der winkelbasierten bzw. merkmalbasierten Distanz zwischen den beiden entsprechenden Dreiecken. Dijkstras Algorithmus zur Bestimmung kürzester Wege muss nun noch etwas erweitert werden, so dass für jedes neu erreichte Dreieck ein Verweis auf den aktuell betrachtete Seed-Knoten gespeichert wird, wenn die Entfernung zu dem Seed-Knoten geringer ist als alle zuvor berechneten Entfernungen. Darüber hinaus sind auch folgende *Regeln für die Patch-Berechnung* zu berücksichtigen:

Regel 1 *Die Seed-Knoten werden in aufsteigender Reihenfolge der Gewichtswerte verarbeitet.*

Regel 2 *Wenn ein Knoten erreicht wird, für den bereits eine geringere gewichtete Distanz zu einem anderen Seed-Knoten ermittelt worden ist, dann wird dieser Knoten im aktuellen Durchlauf des Algorithmus von Dijkstra nicht weiter berücksichtigt.*

Bei Anwendung dieser beiden Regeln wird nicht für sämtliche Knoten des Dualgraphen eine gewichtete Distanz zu allen Seed-Knoten berechnet. Insbesondere müssen nicht alle Knoten des Dualgraphen dem Seed-Knoten zugeordnet sein, dem sie gemäß des gewichteten Distanzmaßes am nächsten liegen. Durch die beiden Regeln ist sichergestellt, dass jedes berechnete Patch zusammenhängend ist. Ohne Veränderung der Seed-Punkte oder anderer Parameterwerte, die zu dem Ergebnis aus Abbildung 5.5 geführt haben, ergibt sich nun das in Abbildung 5.8 dargestellte Ergebnis. Dort ist keine rote „Insel“ mehr vorhanden und beide Patches sind damit zusammenhängend.

Ein Pseudocode des Algorithmus zur Patch-Berechnung ist in Listing 5.1 angegeben. Zur besseren Verständlichkeit ist dort auf die Dualraum-Syntax verzichtet worden. Der Ausdruck $\frac{1}{\gamma_{seed}} \cdot d'(t_{akt}, t_{nachbar})$ in Zeile 29 entspricht einer gewichteten winkelbasierten bzw. gewichteten merkmalbasierten Distanz zweier benachbarter Dreiecke bezüglich

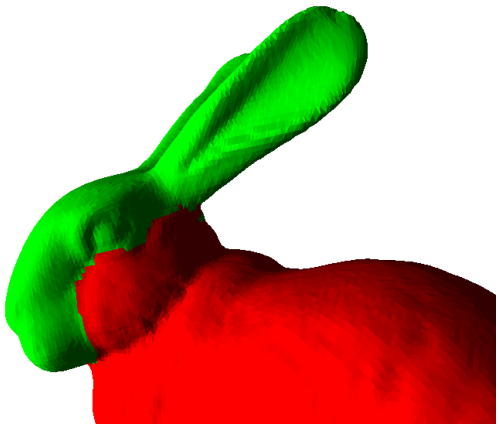


Abbildung 5.8: *Bei Berücksichtigung der beiden Regeln entsteht nun kein nicht-zusammenhängendes Patch mehr.*

des aktuell gewählten Seed-Punktes. Die Seed-Anzahl ist üblicherweise durch einen konstanten Wert begrenzt, so dass die Zeitkomplexität des durch den Pseudocode in Listing 5.1 beschriebenen Algorithmus bei Verwendung einer Priority-Queue $O(m \log n)$ beträgt (vgl. [CLRS01, S. 599]), wobei n die Anzahl der Dualgraph-Knoten, also die Anzahl der Netzdreiecke, und m die Anzahl der Dualgraph-Kanten bezeichnen. Da jedes Dreieck drei Kanten hat, ist die Anzahl der Kanten des Dreiecksnetzes und damit auch die Anzahl der Dualgraph-Kanten durch $m \leq 3n$ beschränkt. Dies führt zu einer Zeitkomplexität von $O(n \log n)$.

5.2.3.2 Patch-Berechnung bei Seed-Clustern

In dieser Arbeit wurden bereits verschiedene Arten von Seed-Clustern angesprochen. So gibt es Seed-Linien, die manuell gezogen werden, aber auch die Möglichkeit, mehrere einzelne Seed-Punkte zu setzen, die zusammen ein Patch definieren, ohne dabei zwangsläufig direkt nebeneinander zu liegen brauchen. Eine weitere Art von Seed-Cluster wird mit den Satelliten-Seeds in Kapitel 6 eingeführt. Seed-Cluster unterscheiden sich von normalen Seeds nur insofern, als dass mehrere Seeds ein Patch definieren. Dabei definiert jeder dieser Seeds einen Patch-Ausschnitt, der im Folgenden *Sub-Patch* genannt wird. Ein Patch ist somit die Vereinigung aller Sub-Patches, die von den Elementen eines Seed-Clusters definiert werden.

Seed-Gewichte wurden eingeführt, um den Einfluss einzelner Seed-Punkte gezielt anpassen zu können. Dahinter verbirgt sich die Idee, einigen Seeds mehr Bedeutung zukommen zu lassen als anderen. Dies gilt aber nicht nur für „normale“ Seeds; auch die Gewichte der Seeds eines Clusters können durchaus voneinander abweichen. Dadurch wird ermöglicht, einen lokal begrenzten Einfluss auf das Patch des Seed-Clusters zu nehmen. Dies spielt insbesondere bei dem im folgenden Kapitel vorgestellten Satelliten-Seeding eine zentrale Rolle. Im Gegensatz zu Satelliten-Seeds und anderen Seed-Clustern sollen alle Elemente einer Seed-Linie stets denselben Gewichtswert haben.

```
1  Patchberechnung(Dreiecksnetz M,
2                  Seed-Dreieck-Liste L_seed,
3                  Distanzfunktion d' ∈ {d^ζ_ang, d^{ζ,η}_feature})
4  {
5     // Initialisierung:
6     T:=Menge aller Dreiecke aus M;
7     for each t ∈ T do
8     {
9        dist[t] := ∞;
10    }
11
12    // Sortieren der Seed-Dreiecke nach Seed-Gewichten:
13    L_sortiert:=sortiereNachAufsteigendenGewichtswerten(L_seed);
14
15    for i := 1 to L_sortiert.size() do
16    {
17       L:=leere Liste;
18       t_seed := L_sortiert[i];
19       dist[t_seed] := 0;
20       γ_seed:=Seed-Gewicht von t_seed;
21       füge t_seed in L ein;
22
23       while |L| > 0 do
24       {
25          wähle t_akt ∈ {t ∈ L | dist[t] ist minimal};
26          entferne t_akt aus L;
27          for each benachbartes Dreieck t_nachbar von t_akt do
28          {
29             tmpdist := dist[t_akt] + 1/γ_seed · d'(t_akt, t_nachbar);
30             if (tmpdist < dist[t_nachbar])
31             {
32                dist[t_nachbar] := tmpdist;
33
34                // Anpassen der Patch-Zugehörigkeit:
35                Patch-Nummer von t_nachbar := Patch-Nummer von t_seed;
36                füge t_nachbar in L ein;
37             }
38          }
39       }
40    }
41 }
```

Listing 5.1: *Pseudocode des Algorithmus zur Patch-Berechnung.*

Der in Abschnitt 5.2.3.1 vorgestellte Algorithmus zur Patch-Berechnung ist so allgemein gehalten, dass auch Seed-Cluster verarbeitet werden können; cluster-spezifische Anpassungen sind also nicht notwendig. Auf Seite 61 wurden zwei zum Algorithmus gehörende Regeln angegeben. Diese besagen, dass die Verarbeitung der Seeds in aufsteigender Reihenfolge der Gewichtswerte geschehen muss und dass nicht für alle Seed-Punkte ein vollständiger Durchlauf des Dualgraphen erfolgen darf. Trotz der Gruppierung der Seeds zu Clustern haben diese Regeln auch hier Bestand. Demnach werden mitunter nicht alle Seeds eines Clusters direkt aufeinanderfolgend verarbeitet, sondern Seeds anderer Patches zwischendurch berücksichtigt. Andernfalls könnten auch bei Verwendung von Seed-Clustern – ähnlich wie zuvor für einzelne Seeds beschrieben – Patches auftreten, die nicht zusammenhängend sind.

5.3 Interaktionsmöglichkeiten

Die Verwendung von Seed-Punkten ermöglicht es den Anwendenden, vergleichsweise intuitiv Einfluss auf die Gestalt der Patches zu nehmen. Wenn die berechneten Patches nicht den Vorstellungen entsprechen, haben die Anwendenden verschiedene Interaktionsmöglichkeiten. Diese unterteilen wir in zwei Klassen: Interaktionsmöglichkeiten, die eine erneute Patch-Berechnung erfordern, und solche, bei denen das nicht der Fall ist. Erstere werden hier *patchverändernde Interaktionen* genannt, letztere *patcherhaltende Interaktionen*. Zu den patcherhaltenden Interaktionen zählen folgende Eingriffsmöglichkeiten:

1. **Gruppieren von Patches:** Einzelne Patches können zu einer so genannten *Patch-Gruppe* zusammengefasst werden. Eine Patch-Gruppe muss stets ein zusammenhängendes Teilnetz bilden, so dass sie auch als ein großes Patch angesehen werden kann, das mehrere Seed-Punkte enthält. Die Erzeugung von Patch-Gruppen kann auf unterschiedliche Weisen erfolgen: Es ist möglich, mehrere Patches direkt zu einer Patch-Gruppe zusammenzufassen oder bestehende Patch-Gruppen zu einer größeren zu verschmelzen. Außerdem können bereits bestehende Patch-Gruppen um weitere Patches erweitert werden. Alle Informationen über die zu einer Patch-Gruppe gehörenden Patches sowie deren Seed-Punkte bleiben trotz Gruppierung erhalten.

2. **Auflösen von Patch-Gruppen:** Jede aus mehreren Patches bestehende Patch-Gruppe kann wieder in ihre Bestandteile, also die einzelnen Patches, zerlegt werden.

Diese beiden patcherhaltenden Interaktionen haben keine Auswirkung auf Patch-Grenzen bzw. auf die Zuordnung der Dreiecke zu den einzelnen Seed-Punkten. Somit besteht keine Notwendigkeit, eine erneute Patch-Berechnung durchzuführen, sofern ansonsten keine patchverändernde Interaktion erfolgt ist. Patchverändernde Interaktionen erfordern hingegen eine Neuberechnung der Patches, da diese Interaktion zu anderen Patch-Formen führen. Beim EvOpSeg-Verfahren sind folgende patchverändernde Interaktionen vorgesehen:

1. **Hinzufügen weiterer Seed-Punkte:** Durch das Hinzufügen weiterer Seed-Punkte werden neue Patches definiert, die Teilstücke einiger bisheriger Patches ersetzen. Auf diese Weise lassen sich in manchen Situationen bessere Grenzen und damit auch natürlichere Patches bzw. Patch-Gruppen erzeugen. Das Gewicht eines jeden neu eingefügten Seeds wird mit dem Standard-Wert für Seed-Gewichte initialisiert; beim `EvOpSeg-Tool` ist der Wert 1.0 dafür gewählt worden.

2. **Löschen einzelner Seed-Punkte:** Das Löschen eines Seed-Punktes bzw. des zugehörigen Patches P wirkt sich lediglich auf die direkt um P herum liegenden Patches aus, die dadurch entsprechend vergrößert werden. Alle übrigen Patches bleiben davon unbeeinflusst.

3. **Verschieben von Seed-Punkten:** Die Anwendenden können Einfluss auf die Lage eines Patches nehmen, indem sie dessen Seed-Punkt verschieben. Gehört ein solches Patch einer Patch-Gruppe an, dann darf der Seed-Punkt nur so verschoben werden, dass die Patch-Gruppe auch im Anschluss noch zusammenhängend ist.
   Das Verschieben eines einzelnen Elements einer Seed-Linie führt im Allgemeinen dazu, dass die Seed-Linie nicht länger zusammenhängend ist, also keinen Dreieckszug mehr bildet. Werden mehrere Seeds derart verschoben, ist oft sogar kein Liniencharakter mehr zu erkennen; vielmehr handelt es sich danach stattdessen um ein gewöhnliches Seed-Cluster. Dies ist zwar keineswegs problematisch, widerspricht jedoch ein wenig dem Gedanken von Seed-Linien aus Abschnitt 5.1.1. Eine Verschiebung der kompletten Seed-Linie erscheint sinnvoller. Aufgrund der Struktur von Dreiecksnetzen kann sich dabei die Gestalt der Linie sowie ihre Länge ändern, global gesehen bleibt ihr Verlauf in etwa gleich, sofern die Verschiebung nicht allzu groß ist.

4. **Ändern der Gewichtswerte:** Die Gewichtswerte der einzelnen Seed-Punkte können unabhängig voneinander verändert werden. Die Änderung eines einzigen Gewichtswertes wirkt sich entsprechend auf den Einfluss des Seeds aus. Bei der Änderung mehrerer Gewichtswerte ist hingegen das Verhältnis der Werte zueinander für die genaue Auswirkung entscheidend.

5. **Ändern von ζ bzw. η:** Die Anwendenden können durch Variation der Werte von ζ und η den Einfluss des dihedralen Winkels bzw. des merkmalbasierten Anteils bei der Distanzberechnung verändern.

Wie oben erwähnt muss bei patchverändernden Interaktionen eine neue Patch-Ermittlung erfolgen. Es ist jedoch nicht in allen Fällen notwendig, sämtliche Patches neu zu berechnen; manchmal reicht auch eine *inkrementelle Patch-Aktualisierung* aus, bei der nur einige der Patches zu betrachten sind. Damit eine inkrementelle Aktualisierung möglich ist, muss für jede Dreiecksfacette deren bei der vorherigen Patch-Berechnung ermittelter Abstand zum nächstgelegenen Seed-Punkt bekannt sein. Dies bedeutet, dass nun nicht für alle Dreiecke, sondern nur noch für die der betroffenen Patches bzw. Sub-Patches, eine Initialisierung der Abstände mit dem Distanzwert ∞ erfolgen darf; die Zeilen 7 bis 10 des Pseudocodes aus Listing 5.1 sind dafür also entsprechend zu ersetzen. Eine inkrementelle Patch-Aktualisierung kann in folgenden Situationen erfolgen:

- **Situation 1:** Es sind neue Seed-Punkte hinzugefügt worden, ohne dass weitere patchverändernde Interaktionen erfolgt sind. Bei der Patch-Berechnung müssen nur die neuen Seed-Punkte sowie alle solchen berücksichtigt werden, deren Gewichtswert größer ist als der kleinste Gewichtswert aller neu hinzugefügter Punkte. Wenn alle Seed-Gewichte den gleichen Wert haben oder die Gewichtswerte der neu eingefügten Seeds größer sind als die der anderen, reicht es sogar aus, lediglich die neu hinzugefügten Seed-Punkte zu betrachten.

- **Situation 2:** Es sind Seed-Punkte gelöscht worden, ohne dass weitere patchverändernde Interaktionen stattgefunden haben. Da jeder Seed-Punkt ein Patch oder im Falle eines Seed-Clusters ein Sub-Patch induziert, wirkt sich das Löschen lediglich auf die angrenzenden Patches bzw. Sub-Patches aus. Somit reicht es aus, auf dem Teilnetz, das nur diese angrenzenden sowie die zu den gelöschten Seed-Punkten gehörenden Patches bzw. Sub-Patches enthält, eine erneute Patch-Berechnung durchzuführen. Das gleiche Ergebnis lässt sich auch erzielen, indem auf dem kompletten Oberflächennetz eine neue Patch-Berechnung für alle Seed-Punkte durchgeführt wird, deren Seed-Gewicht mindestens so groß ist wie das kleinste Gewicht aller zu den angrenzenden Patches bzw. Sub-Patches gehörenden Seed-Punkte.

- **Situation 3:** Seed-Punkte sind verschoben worden, ohne dass weitere patchverändernde Interaktionen erfolgt sind. Da das Verschieben eines Seeds auch als Löschen und anschließendem Hinzufügen eines neues Seeds (mit denselben Patch-Zugehörigkeiten und sonstigen Eigenschaften, die der gelöschte Seed hatte) angesehen werden kann, ist hier ebenfalls eine inkrementelle Patch-Aktualisierung möglich. Wie beim Löschen von Seeds werden alle Patches bzw. Sub-Patches betrachtet, die an ein gelöschtes Patch oder Sub-Patch angrenzen. Es sei γ_{min} das minimale Gewicht aus der Menge aller zugehörigen Seeds zuzüglich aller zu verschiebenden Seeds. Die Seed-Verschiebung wird durchgeführt und anschließend für alle Seeds, deren Gewichtswert mindestens γ_{min} beträgt, eine Patch-Berechnung durchgeführt.

- **Situation 4:** Seed-Gewichte sind verändert worden, ohne dass weitere patchverändernde Interaktionen erfolgt sind. Bei der Patch-Berechnung müssen dann nur die Seed-Punkte berücksichtigt werden, deren Seed-Gewicht mindestens so groß ist wie das kleinste aller angepassten Gewichte.

Auch bei beliebigen Kombinationen dieser vier Situationen ist prinzipiell eine inkrementelle Patch-Aktualisierung möglich. Die Bestimmung des Gewichtswertes, über den geregelt wird, für welche Seed-Punkte eine erneute Patch-Berechnung zu erfolgen hat, ist dann entsprechend anzupassen, so dass das Minimum aller in Frage kommenden Werte genommen wird. Auf keinen Fall ist jedoch eine inkrementelle Patch-Aktualisierung möglich, wenn die globalen Parameter ζ bzw. η verändert werden. Eine solche Änderung ist nämlich unabhängig von den Seed-Punkten und wirkt sich auf alle Patches gleichermaßen aus.

Der Prozess des interaktiven Eingreifens mit anschließender erneuter Patch-Berechnung kann beliebig oft wiederholt werden. Auf diese Weise ist es den Anwendenden möglich, das Ergebnis immer weiter an ihre Erwartungen anzunähern.

5.4 Ergebnisse und Diskussion

In diesem Abschnitt werden zunächst die Qualität des automatischen Seeding-Verfahrens sowie die Auswirkungen von allgemeinen Seed-Clustern und Seed-Linien untersucht. Darüber hinaus erfolgt eine Analyse des Einflusses, den die Wahl der Distanzfunktion auf das Segmentierungsergebnis hat. Abschließend werden einige Ergebnisse der Initialsegmentierung vorgestellt, anhand derer auch diskutiert wird, bei welchen Modellarten mit guten Segmentierungsergebnissen zu rechnen ist und welche Situationen Herausforderungen für das EvOpSeg-Verfahren darstellen.

Analyse des automatischen Seeding

Aufgrund des Verfahrens ist bei „natürlichen“ Modellen zu erwarten, dass der erste automatisch gesetzte Seed am Ende einer Extremität liegt. Diese Erwartung konnte im Rahmen der durchgeführten Experimente bei nahezu allen untersuchten Modellen bestätigt werden. Technische Modelle wie mechanische Bauteile oder Rotationskörper enthalten normalerweise keine entsprechenden Extremitäten. Bei solchen Modellen wird der erste Seed meistens an einem Ende des Modells platziert.

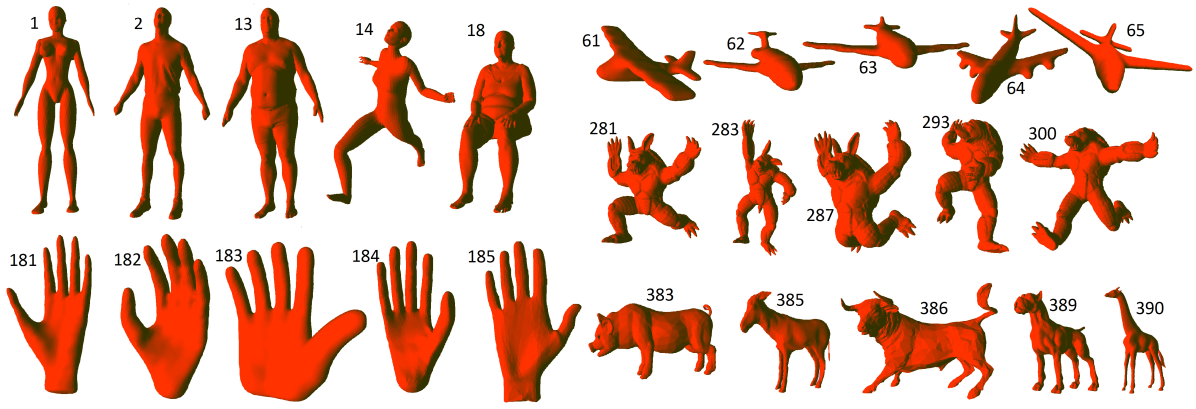


Abbildung 5.9: *Zur Analyse des automatischen Seeding verwendete Modelle.*

Im Folgenden wird genauer untersucht, wie gut sich das automatische Seeding zum Anordnen adäquater Seed-Punkte eignet. Dazu sind für fünf Modellkategorien aus dem Benchmark von Chen et al. [CGF09], auf den in Kapitel 8 noch näher eingegangen wird, jeweils fünf Modelle gewählt und untersucht worden; die verwendeten Modelle sind inklusive der im Benchmark verwendeten Nummern in Abbildung 5.9 zu sehen. Bei den Kategorien handelte es sich um „Menschen“, „Armadillos“, „vierbeinige Tiere“, „Hände“ und „Flugzeuge“. Die Wahl der konkreten Modelle ist so erfolgt, dass diese ein möglichst breites Spektrum abdecken. Beispielsweise sollten sowohl stehende als auch laufende und sitzende Menschenmodelle dabei sein. Alle Modelle einer Kategorie besitzen Komponenten mit der gleichen Semantik. So verfügt jedes vierbeinige Tier über einen Kopf, einen Körper sowie vier einzelne Beine. Die Seeding-Qualität

ist offensichtlich gut, wenn die beiden folgenden Bedingungen erfüllt sind, die hier als *Indikatoren für eine gute Seeding-Qualität* angesehen werden und im weiteren Verlauf als Grundlage für die Analyse des automatischen Seeding dienen:

Indikator 1 *Jede Komponente enthält einen Seed-Punkt, ohne dass eine zu starke Übersegmentierung vorliegt. Im Idealfall enthält jede Komponente genau einen Seed, so dass es also keine Übersegmentierung gibt.*

Indikator 2 *Jeder Seed-Punkt liegt zentral innerhalb der Komponente.*

Jedes Modell der Kategorie „Hände“ besteht aus einem Daumen, einem Zeigefinger, einem Mittelfinger, einem Ringfinger, einem kleinen Finger, der eigentlichen Handfläche und gegebenenfalls noch einem Handgelenk. In Tabelle 5.1 ist für jede Komponente angegeben, wie viele Seed-Punkte mit dem automatischen Verfahren platziert werden müssen, damit sie einen Seed enthält. Es ist zu erkennen, dass bei vier Modellen die Anzahl der insgesamt benötigten Seeds mit der Anzahl der Komponenten übereinstimmt. Demnach ist der oben genannte erste Indikator für eine gute Seeding-Qualität offensichtlich erfüllt. Da zudem die Seeds der Finger allesamt auf den Fingerspitzen angeordnet worden sind und die anderen beiden Seeds nicht am Komponentenrand liegen, wird auch dem zweiten Indikator Genüge getan. Lediglich bei dem Modell mit der Nummer 182 wird ein zusätzlicher Seed-Punkt benötigt. Auffällig ist auch, dass in allen fünf Fällen der erste Seed jeweils am Handgelenk platziert wurde und dass tendenziell eher die Finger als die Handfläche einen Seed-Punkt erhalten. Darüber hinaus wird bei einer Übersegmentierung zunächst die Handfläche mit weiteren Seeds versehen. Das in Abschnitt 5.1.2 genannte Ziel, die ersten paar Seeds auf Extremitäten zu setzen bevor weitere Seeds annähernd gleichmäßig über das Modell verteilt werden, ist somit offensichtlich erreicht worden.

Modell-Nr.	Daumen	Zeigefinger	Mittelfinger	Ringfinger	kleiner Finger	Handfläche	Handgelenk
181	4	5	2	3	6	7	1
182	3	4	7	2	8	5	1
183	6	4	2	5	3	7	1
184	5	4	2	6	3	7	1
185	3	5	2	6	4	7	1

Tabelle 5.1: *Auswertung des automatischen Seeding (Handmodelle).*

Ein Flugzeug besteht aus den beiden Tragflächen, dem Rumpf, einer Nase und dem Heck. Der Tabelle 5.2 ist zu entnehmen, wie viele Seed-Punkte bei dem automatischen

Seeding benötigt werden, um diese Teile als Segmente erhalten zu können. Wie bei den Händen wird auch hier bei lediglich einem Modell eine Übersegmentierung benötigt, damit jede Komponente einen Initial-Seed erhält. Somit entspricht dies dem ersten Indikator für eine gute Seeding-Qualität. Bei allen Tests erhielt der Rumpf stets als letzte Komponente einen Seed-Punkt, während Heck und Flugzeugnase recht früh mit einem Seed versehen wurden. Letzteres hängt offensichtlich damit zusammen, dass die meisten Flugzeugmodelle länger als breit sind. Der Einteilung eines Flugzeugs in die oben genannten Komponenten lag der Gedanke zugrunde, dass die Größe der einzelnen Komponenten nicht allzu stark voneinander abweichen soll. Deswegen ist das Heck zunächst als eine einzige, etwas komplexere Komponente angesehen worden. Wenn die Seed-Punkte so gesetzt werden sollen, dass jeder Heckflügel einen enthält, werden durchschnittlich elf Seeds benötigt. Dies führt dazu, dass bei den größeren Komponenten des Flugzeugs eine leichte Übersegmentierung vorliegt, die allerdings aufgrund der Möglichkeit, Patches zu gruppieren, unproblematisch ist. Von denen zur Übersegmentierung gehörenden Seeds liegt allerdings kein einziger in der Nähe eines Komponentenrandes, so dass auch hier die durch den zweiten Indikator formulierte Bedingung erfüllt ist.

Modell-Nr.	linke Tragfläche	rechte Tragfläche	Rumpf	Flugzeugnase	Heck
61	2	1	8	4	3
62	3	4	5	1	2
63	3	4	5	2	1
64	4	3	5	1	2
65	3	4	5	1	2

Tabelle 5.2: *Auswertung des automatischen Seeding (Flugzeugmodelle).*

Als nächstes werden vierbeinigen Tiere betrachtet, die einen Schwanz besitzen. Der Tabelle 5.3 kann man entnehmen, dass als erstes der Kopf oder der Schwanz einen Seed-Punkt erhält. Auf dem Rumpf wird hingegen wie bei den Flugzeugmodellen erst recht spät ein solcher platziert. In allen untersuchten Fällen reichen sieben Seed-Punkte aus, um auf jeder der sieben Komponenten einen Seed mit dem automatischen Seeding-Verfahren zu platzieren. Die Seeds der Beine werden bis auf wenige Ausnahmen direkt unter den Füßen gesetzt. Dies ist vergleichbar mit dem Verhalten des Seeding-Algorithmus, die Seed-Punkte der Finger auf den Fingerspitzen zu platzieren. Auch bei diesen vierbeinigen Tieren werden die Seeds gemäß der beiden Indikatoren recht plausibel angeordnet.

Die Modelle der Kategorie „Menschen“ weisen zu denen der Kategorie „Armadillos“ gewisse Ähnlichkeiten auf. Im Wesentlichen besitzen sie Komponenten mit gleicher

Modell-Nr.	Kopf	Rumpf	linkes Vorderbein	rechtes Vorderbein	linkes Hinterbein	rechtes Hinterbein	Schwanz
383	2	6	3	7	4	5	1
385	2	7	3	6	5	4	1
386	4	7	2	5	6	3	1
389	1	7	4	5	2	3	6
390	1	7	3	5	4	2	6

Tabelle 5.3: *Auswertung des automatischen Seeding (vierbeinige Tiere).*

Semantik (vgl. Tabellen 5.4 und 5.5). Allerdings verfügen einige der untersuchten Armadillos zusätzlich über einen Schwanz; bei den anderen Armadillomodellen ist in Tabelle 5.5 durch „n.v.“ angegeben, dass ein Schwanz nicht vorhanden ist. Anders als bei den zuvor geschilderten Untersuchungen enthalten die Tabellen 5.4 und 5.5 nun Komponenten einer feineren Granularität: Aufgrund der klar erkennbaren Hände und Füße sollten sowohl diese als auch die Arme und Beine jeweils einen Seed-Punkt erhalten. Eine Ausnahme bildet lediglich das Modell mit der Nummer 18. Es handelt sich um das Modell einer sitzenden Frau, die ihre Hände auf die Oberschenkel gelegt hat. Dadurch stellen die Hände keine abstehenden Extremitäten dar und werden bei einer noch sinnvollen Anzahl von Seeds nicht mit einem solchen versehen. Entsprechend sind in Tabelle 5.4 dafür auch keine Werte angegeben. Während bei allen Armadillo- sowie Menschenmodellen (abgesehen von Nummer 18) Füße und Hände recht früh einen Seed-Punkt erhalten, ist dies bei den Armen und Beinen erst deutlich später der Fall. Entsprechend müssen sowohl bei Menschen- als auch bei Armadillomodellen durchschnittlich rund 13 Seeds genutzt werden, um jede Komponente mit einem Seed zu versehen. Während dies bei den Menschenmodellen eine leichte Übersegmentierung bedeutet, liegt der Wert bei Armadillos näher an der Anzahl der Komponenten. Auch bei diesen beiden Modellkategorien werden die ersten Seeds tendenziell eher auf Extremitäten bzw. meist sogar an deren Enden platziert.

Wie oben beschrieben platziert das automatische Seeding-Verfahren primär Seed-Punkte an den Enden von Extremitäten. Dies funktioniert aber nur dann zuverlässig, wenn die Extremitäten im Verhältnis zum gesamten Modell nicht zu klein sind, d.h. wenn sie weit genug hervorstehen. Die Ohren eines Pferdes werden beispielsweise erst äußerst spät mit einem Seed versehen – beim Modell 387 aus dem Benchmark von Chen et al. [CGF09] waren 99 Seeds notwendig, damit das erste Ohr einen erhielt. Somit produziert das automatische Seeding-Verfahren nicht für jedes Modell und jede gewünschte Segmentierungsgranularität eine sinnvolle Seed-Anordnung, die ausschließlich zu Segmenten mit semantischer Bedeutung führt. Während bei den Flugzeugmodellen in den Fällen, wenn auch jede Komponente des Heckflügels einen Seed erhalten

Modell-Nr.	Kopf	Rumpf	linker Arm	rechter Arm	linke Hand	rechte Hand	linkes Bein	rechtes Bein	linker Fuß	rechter Fuß
1	5	8	9	19	4	2	6	7	1	3
2	1	6	11	12	5	4	10	8	3	2
13	1	6	12	13	5	4	8	10	3	2
14	6	5	11	10	2	4	8	9	3	1
18	1	4	9	12	-	-	7	8	3	2

Tabelle 5.4: *Auswertung des automatischen Seeding (Menschenmodelle).*

Modell-Nr.	Kopf	Rumpf	linker Arm	rechter Arm	linke Hand	rechte Hand	linkes Bein	rechtes Bein	linker Fuß	rechter Fuß	Schwanz
281	6	7	13	11	3	2	10	9	1	4	5
283	6	7	13	11	4	2	8	9	1	3	5
287	5	6	8	9	1	3	12	11	4	2	10
293	5	6	17	10	4	2	13	9	1	3	n.v.
300	5	6	9	10	3	2	7	8	1	4	n.v.

Tabelle 5.5: *Auswertung des automatischen Seeding (Armadillomodelle).*

soll, lediglich eine leichte Übersegmentierung entsteht, liegt bei dem Pferde-Beispiel eine nicht mehr zu akzeptierende Übersegmentierung vor.

Auswirkungen der beiden Distanzfunktionen

Mit der gewichteten winkelbasierten Distanz und der gewichteten merkmalbasierten Distanz sind zwei Abstandsmaße für die Patch-Berechnung definiert worden. Dabei stellt die merkmalbasierte Distanz eine Erweiterung der winkelbasierten Distanz dar. Abbildung 5.10 ist zu entnehmen, dass die merkmalbasierte Distanz bei manchen Formen durchaus zu einem besseren Ergebnis führt als die winkelbasierte. Als „perfekt“ ist auch die Segmentierung im rechten Teilbild sicherlich nicht zu bezeichnen. Die in den nachfolgenden Kapiteln beschriebenen Techniken des Satelliten-Seeding sowie der evolutionären Patch-Optimierung bieten allerdings in solchen Fällen Abhilfe, indem sie eine gezielte Verbesserung der Patch-Grenzen ermöglichen.

Bei Untersuchungen, die im Rahmen dieser Arbeit durchgeführt worden sind, hat sich manchmal die winkelbasierte Distanz und manchmal die merkmalbasierte Distanz als vorteilhaft erwiesen. Die Unterschiede waren jedoch zumindest für die gewählten Werte von ζ und η häufig eher gering[3]. Deswegen ist für die folgenden Beispiele dieses Abschnitts die winkelbasierte Distanz verwendet worden.

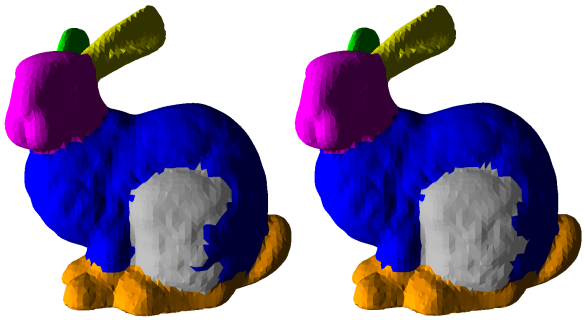


Abbildung 5.10: *Die unterschiedliche Auswirkung der gewichteten winkelbasierten Distanz (links) und der gewichteten merkmalbasierten Distanz (rechts) ist insbesondere am grauen Patch zu erkennen.*

Seed-Cluster

Seed-Cluster sind hilfreich, um Segmente zu beschreiben, die in eine Richtung deutlich länger sind als in die anderen Richtungen oder die starke konkave Strukturen beinhalten. Der zweite Punkt hängt mit dem winkelbasierten Anteil der Distanzfunktion zusammen, die bei der Patch-Berechnung zum Einsatz kommt. Ein Beispiel für eine

[3] Auf eine geeignete Wahl wird in Kapitel 8 im Rahmen der evolutionären Patch-Optimierung eingegangen.

längliche Komponente, bei der ein Seed-Cluster hilft, ist in Abbildung 5.11 zu sehen. Im linken Teilbild besitzt der Körper des Drachen lediglich einen Seed-Punkt. Dies führt dazu, dass bei der Patch-Berechnung Teile des Körpers dem Kopf und dem Vorderbein zugeordnet werden. Im rechten Teilbild ist für den Körper ein weiterer Seed angegeben worden, so dass nun also ein Seed-Cluster vorliegt. Dies führt ohne weitere Änderungen zu der abgebildeten Segmentierung.

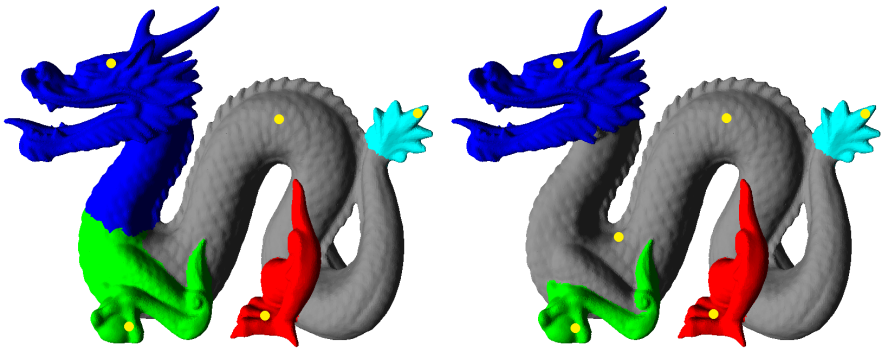


Abbildung 5.11: *Im Gegensatz zum linken Teilbild ist rechts ein Seed-Cluster zur Definition des grauen Patches verwendet worden. Die einzelnen Seeds sind gelb markiert.*

Seed-Linien stellen eine gute Möglichkeit dar, mit geringem manuellem Aufwand komplexe Komponentengrenzen zu beschreiben. Dazu muss auf beiden Seiten in der Nähe der Grenze jeweils eine Seed-Linie gezogen werden. Die Seed-Linien brauchen dabei keineswegs geschlossen zu sein. Im rechten Teilbild der Abbildung 5.12 ist ein Beispiel für solche Seed-Linien zu sehen. Bei den gewählten Parameterwerten (alle Gewichtswerte haben den initialen Wert 1.0) führt dies zu einem plausiblen Ergebnis, bei dem der Kopf des Pferdes zusammen mit dem Hals ein Segment und der restliche Teil ein zweites Segment bilden. Im linken Teilbild existiert zu jedem Patch lediglich genau ein Seed-Punkt. Die Grenze zwischen den Patches ist in dem Fall nicht so akkurat wie bei den Seed-Linien. Um einen besseren Übergang zu erhalten, sind im mittleren Teilbild weitere Seed-Punkte im Bereich der nicht-optimalen Patch-Grenze hinzugefügt worden, so dass sich zwei Seed-Cluster ergeben. Während die Grenze auf der sichtbaren Seite des Pferdemodells nun wesentlich natürlicher aussieht, hat sie sich auf der Rückseite durch das Einfügen der neuen Seed-Punkte deutlich verschlechtert. Dies ist in der Abbildung daran zu erkennen, dass die beiden rechten Beine nun ebenfalls als zum Kopf gehörig segmentiert worden sind. Durch sukzessive Hinzunahme weiterer Seeds auf der Rückseite lässt sich das Problem sicherlich auch in den Griff bekommen, das einmalige Auftragen zweier Seed-Linien auf das Modell ist jedoch weniger aufwendig als ein mehrmaliges Hinzufügen einzelner Seeds zu den Clustern jeweils gefolgt von einer Patch-Berechnung. Bei dem Pferdemodell aus Abbildung 5.12 besitzt zudem die Grenze, die sich durch die Seed-Linien ergeben hat, einen natürlicheren Verlauf als im Falle von allgemeinen Seed-Clustern. Insgesamt scheinen Seed-Linien ein gute Möglichkeit zu sein, in schwierigen Situationen mit vergleichsweise wenig Aufwand gute Ergebnisse

zu erzielen. In einfacheren Fällen reichen jedoch häufig auch allgemeine Seed-Cluster aus.

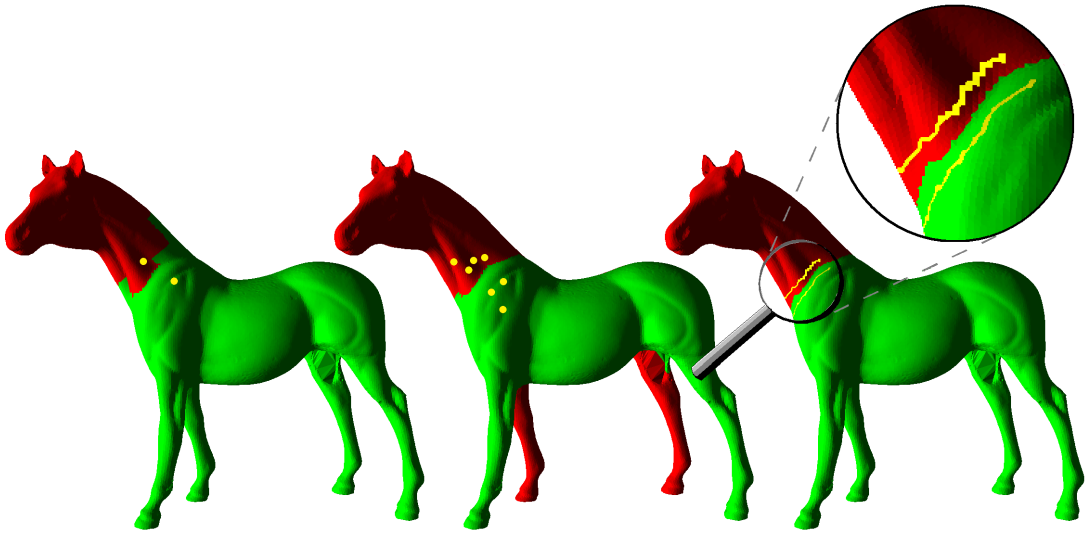


Abbildung 5.12: *„Normale“ Seeds (links), allgemeine Seed-Cluster (Mitte) und Seed-Linien (rechts).*

Weitere Beispielsegmentierungen

In den Abbildungen 5.10 bis 5.12 sind bereits Initialsegmentierungen vorgestellt worden, anhand derer spezielle Aspekte veranschaulicht worden sind. Weitere unter Verwendung der winkelbasierten Distanz erlangte Beispielsegmentierungen sind in Abbildung 5.13 zu sehen. Auf der Teekanne im linken Teilbild sind vier Seeds nach dem in Abschnitt 5.1.2 vorgestellten automatischen Seeding-Verfahren angeordnet worden. Die sich daraus ergebenden Patches entsprechen absolut den Erwartungen, wodurch neben der eigentlichen Patch-Berechnung auch die automatische Anordnung von Seed-Punkten untermauert wird.

Abbildung 5.13: *Initialsegmentierungen unterschiedlicher Modelle.*

Beim Vogel im zweiten Teilbild der Abbildung 5.13 ist ein hybrides Seeding zum Einsatz gekommen: Zunächst sind drei Seed-Punkte automatisch auf dem Modell angeordnet worden, anschließend ist jeder Fuß manuell mit einem weiteren versehen worden. Die sich ergebenden Patches beschreiben mit den Flügeln, den Beinen und dem Körper semantisch-bedeutungsvolle Objektteile. Wie im ersten Teilbild sind auch hier die Grenzen zwischen den Patches sehr gut.

Auf dem im dritten Teilbild gezeigten Krug sind drei Seed-Punkte automatisch angeordnet worden. Jede der drei logischen Komponenten ist dabei mit einem Seed-Punkt versehen worden. Während die Bodenplatte korrekt segmentiert worden ist, läuft allerdings das den Henkel beschreibende Patch im oberen Bereich deutlich zu weit in die zentrale Komponente hinein. Hier liefert die Patch-Berechnung also kein zufriedenstellendes Ergebnis.

Ebenfalls nicht zufriedenstellend sind die im vierten Teilbild dargestellten Patches. Das grüne Patch erstreckt sich deutlich zu weit in die Handinnenfläche hinein. Das den Mittelfinger beschreibende gelbe Patch verläuft sogar auf der hinten liegenden Seite der Hand herum bis unterhalb des kleinen Fingers.

5.5 Fazit

In diesem Kapitel wurde die Initialsegmentierung beschrieben, d.h. die Ermittlung einer geeigneten Zerlegung des gegebenen dreidimensionalen Modells in Patches. Der vorgeschlagene Algorithmus zur Berechnung der Patches berücksichtigt neben geodätischen Distanzen auch die Krümmung des Modells sowie gegebenenfalls gewisse lokale Formmerkmale. Er basiert auf Seed-Punkten, die sowohl automatisch als auch manuell auf der Oberfläche des Modells angeordnet werden können. Das Verhalten des automatischen Seeding wurde für Modelle aus fünf unterschiedlichen Kategorien mit Hilfe zweier Indikatoren für eine gute Seeding-Qualität analysiert. Dabei hat sich herausgestellt, dass die vorgeschlagene Variante der automatischen Seed-Vergabe für die ausgewählten Modelle als recht plausibel anzusehen ist. Sie ist allerdings für eher kurze Extremitäten nicht gut geeignet; derartige Komponenten werden in der Regel nicht mit Seeds versehen. Als problematisch kann sich das automatische Seeding außerdem darstellen, wenn eine Übersegmentierung gewünscht ist, da in einem solchen Fall durchaus Seeds in der Nähe von Komponentengrenzen landen können. Daraus resultieren häufig Patches, die sich über Komponentengrenzen hinweg erstrecken. Auch ein optimales Gruppieren von Patches kann dann zu keinem zufriedenstellenden Ergebnis mehr führen.

Im Gegensatz zum hier vorgestellten automatischen Seeding erlaubt die manuelle Seeding-Variante das gezielte Erzeugen sinnvoller Seed-Cluster. Diese können in Form von allgemeinen Seed-Clustern oder als Seed-Linien vorliegen. Sie führen bei manchen Modellen zu besseren Ergebnissen, beispielsweise wenn einzelne Komponenten des Modells länglich sind oder wenn starke konkave Strukturen innerhalb einer Komponente vorliegen. Dies motiviert die Fragestellung, ob sich auch ein automatisches Seeding

realisieren lässt, welches eigenständig sowohl Seeds als auch Seed-Cluster erzeugt. Dazu wird im folgenden Kapitel ein so genanntes Satelliten-Seeding eingeführt, das eine Möglichkeit darstellt, ausgehend von Initial-Seeds automatisch Seed-Cluster zu erzeugen, welche sich gut für eine automatische oder manuelle Optimierung eignen. Solche Optimierungen oder manuelle Eingriffe sind sinnvoll, da in einigen Fällen die Initialsegmentierung keine optimalen Ergebnisse liefert. Nicht-optimale Initialsegmentierungen sind anhand eines Krugmodells sowie eines Handmodells (Abbildung 5.13) gezeigt worden, die auch in den folgenden Kapiteln erneut aufgegriffen werden.

Kapitel 6

Satelliten-Seeding

Wie im vorangegangenen Kapitel beschrieben, lässt sich die Form der berechneten Patches durch Verschieben der Seed-Punkte oder durch Verändern der Seed-Gewichte beeinflussen. Das im Folgenden vorgestellte Satelliten-Seeding-Verfahren bietet als Erweiterung des normalen Seeding noch bessere bzw. zielgerichtetere Beeinflussungsmöglichkeiten. Voraussetzung ist, dass bereits Seed-Punkte und damit auch zumindest implizit entsprechende Patches vorhanden sind. Beim Satelliten-Seeding werden jedem vorhandenen Seed-Punkt mehrere neue Seeds hinzugefügt, die in der vorliegenden Arbeit aufgrund ihrer Anordnung *Satelliten-Seeds* genannt werden. Zusammengehörende Satelliten-Seeds definieren Patches, die den ursprünglichen recht ähnlich, aber dennoch wesentlich besser anzupassen sind. Damit stellen Satelliten-Seeds eine neuartige Form von Seed-Clustern dar, die im Unterschied zu dem weiter vorne beschriebenen gewöhnlichen Seeds-Clustern aber automatisch erzeugt werden.

Im ersten Abschnitt wird zunächst die grundsätzliche Idee erläutert, bevor in Abschnitt 6.2 ausführlich auf die Ermittlung der konkreten Satelliten-Seeds eingegangen wird. Jeder Satelliten-Seed definiert einen Patch-Ausschnitt, weswegen in dem Zusammenhang von *Sub-Patches* gesprochen wird. Abschnitt 6.3 geht auf die Berechnung der Sub-Patches ein.

6.1 Grundidee

Nach erfolgter Initialsegmentierung können Seed-Punkte verschoben sowie Seed-Gewichte verändert werden, um bessere Patches zu erreichen. Diese Eingriffe wirken sich allerdings nicht nur lokal aus, d.h. der Einfluss ist nicht unbedingt auf einzelne Abschnitte der Patch-Grenze beschränkt. Vielmehr bewirkt das Verschieben eines Seed-Punktes bzw. die Anpassung seines Gewichtswertes in der Regel, dass auch „gute“ Patch-Grenzen verändert werden. Dies wird in Abbildung 6.1 schematisch illustriert. Die Seed-Dreiecke sind gelb dargestellt, die Patches durch andere Farben hervorgehoben. Die Grenze zwischen dem blauen und dem roten Patch entspricht im linken Teilbild absolut den Erwartungen, während die Grenze zwischen dem roten und dem

grünen Patch keineswegs optimal ist. Wenn das Gewicht des Seed-Punktes, der das rote Patch definiert, vergrößert wird, um eine bessere Grenze zum grünen Patch zu erhalten, kann dadurch der andere Rand des Patches negativ beeinflusst werden. Eine solche Situation ist exemplarisch im rechten Teilbild dargestellt.

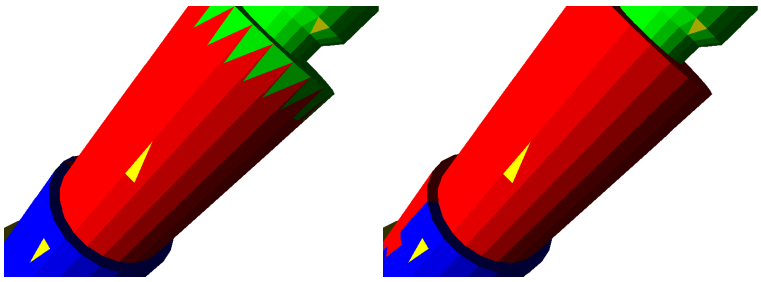


Abbildung 6.1: *Eine Veränderung des Seed-Gewichts kann trotz einer Verbesserung der Grenze zu einem benachbarten Patch dazu führen, dass sich eine andere Grenze verschlechtert (links unten im rechten Teilbild).*

Die zentrale Idee des Satelliten-Seeding ist es, das beschriebene Problem zu umgehen, indem die Ränder der Patches in kleinere Teilstücke unterteilt werden. Erreicht wird dies, indem jedes Patch automatisch durch mehrere kleinere Patches, den Sub-Patches, ersetzt wird. Sub-Patches bieten deutlich mehr Freiheitsgrade bzw. Steuerungsmöglichkeiten: Sie können relativ unabhängig voneinander angepasst und optimiert werden. Für ein Patch P aus der Initialsegmentierung werden durch das Satelliten-Seeding mehrere Sub-Patches ermittelt. Die Vereinigung dieser Sub-Patches wird als neues Patch P' angesehen. P' sollte in etwa mit P übereinstimmen, eine exakte Übereinstimmung ist jedoch nicht gefordert. Allerdings darf P' nicht in einzelne Teile zerfallen, sondern es muss stets zusammenhängend sein. Jedes Sub-Patch wird durch einen gewichteten Seed-Punkt definiert und ist in gleicher Weise steuerbar wie die Patches. Die für die Ersetzung eines Patches durch Sub-Patches benötigten Seed-Punkte werden wie Satelliten um die Initial-Seeds herum angeordnet, weswegen der Begriff *Satelliten-Seeds* gewählt worden ist.

Das Satelliten-Seeding-Konzept kann einerseits als automatisches Pendant zum interaktiven Anlegen von Seed-Clustern angesehen werden. Andererseits stellt es auch eine technische Umsetzung der Idee dar, für einen Initial-Seed mehrere Seed-Gewichte zu definieren, die jeweils in verschiedene Richtungen wirken. Wird der Gewichtswert eines Satelliten-Seed ein wenig verändert, wirkt sich diese Änderung in der Regel nur in eine Richtung aus. So sollen negative Effekte – wie der in Abbildung 6.1 skizzierte – vermieden werden.

Die Ersetzung eines Patches durch mehrere Sub-Patches wird in Abbildung 6.2 veranschaulicht. Im linken Teilbild ist die Grenze zwischen dem roten Rumpf und dem grauen Heck-Bereich des Flugzeugs nicht zufriedenstellend getroffen. Würde die Position oder der Gewichtswert des Seed-Punktes angepasst werden, der das rote Patch definiert, hätte dies auch Auswirkungen auf den hellblauen Bereich an der Flugzeugnase. Eine Anpassung des Seeds, der das graue Patch definiert, könnte hingegen einen zu

weit unten liegenden Rand des rosa Patches (also des Seitenruders) verursachen. Die Ersetzung des roten Patches durch vier logische Sub-Patches im rechten Teilbild stellt eine elegante Lösung des Problems dar: Eine moderate Erhöhung des Seed-Gewichts für das dunkelblaue Patch wirkt sich in keiner Weise auf den Rand der hellblauen Flugzeugnase aus, da die anderen Sub-Patches als eine Art „Puffer“ wirken. Die Darstellung der Sub-Patches mit vier unterschiedlichen Farben ist in Abbildung 6.2 lediglich zur Veranschaulichung der Grundidee erfolgt; sinnvollerweise wird den Anwendenden das gesamte Patch und nicht die einzelnen Sub-Patches präsentiert.

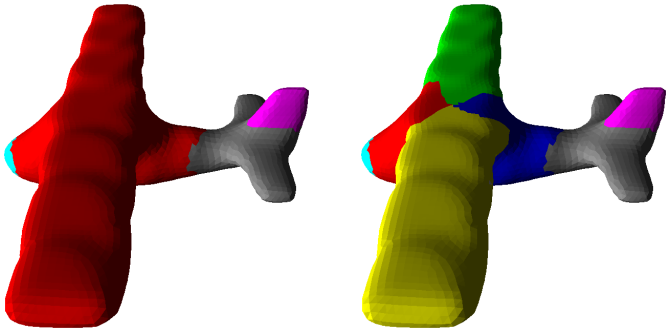


Abbildung 6.2: *Schematische Veranschaulichung der Sub-Patches: Das rote Patch aus dem linken Teilbild ist im rechten Teilbild durch vier Sub-Patches ersetzt worden.*

6.2 Automatisches Platzieren von Satelliten-Seeds

In Kapitel 5 sind unterschiedliche Arten von Seed-Punkten definiert worden. Neben normalen Seed-Punkten, die jeweils ein Patch definieren, gibt es beim EvOpSeg-Verfahren auch Seed-Cluster. Seed-Linien stellen eine spezielle Seed-Cluster-Art dar. Sie zeichnen sich dadurch aus, dass jedes Element der Seed-Linie mindestens ein weiteres Seed-Dreieck als direkten Nachbarn hat. Seed-Linien können insbesondere zur genauen Detektion von Komponentengrenzen genutzt werden, indem jeweils eine auf jeder Seite der gewünschten Grenze platziert wird. Das Hinzufügen zusätzlicher Satelliten-Seeds erscheint in solchen Fällen nicht besonders sinnvoll, da die Seed-Linien bereits recht nahe an der Grenze liegen. Je nach Abstand könnten dann Satelliten-Seeds durchaus auch jenseits der Grenze liegen, was offensichtlich für die Qualität der Patches nicht förderlich ist. Darüber hinaus wäre auch zu erwarten, dass bei Anwendung des hier vorgestellten Satelliten-Seeding-Verfahrens eine Kollision zwischen Satelliten-Seeds und Elementen der Seed-Linie stattfindet. Aus den genannten Gründen bleiben Seed-Linien beim Satelliten-Seeding unberücksichtigt. Alle anderen Seed-Punkte werden hingegen berücksichtigt, auch wenn sie zu sonstigen Seed-Clustern gehören.

Zunächst wird für jeden Seed ein Bereich – der so genannte *Orbit* – ermittelt, auf dem die Satelliten-Seeds anzuordnen sind. Erst wenn für alle Seeds Orbits bestimmt

worden sind, werden diese mit Satelliten-Seeds versehen. Im Folgenden wird die Bestimmung des Orbits sowie die Anordnung der Satelliten-Seeds anhand eines einzigen Initial-Seed $\mathbf{p}_{seed}$ erklärt. Die zu $\mathbf{p}_{seed}$ gehörenden Satelliten-Seeds werden möglichst gleichmäßig um diesen herum angeordnet. Ein Beispiel dafür ist in Abbildung 6.3 zu sehen. Die Initial-Seeds sind im linken Teilbild gelb hervorgehoben[1]. Im rechten Teilbild sind zusätzlich auch die Satelliten-Seeds zu sehen: Jeder Initial-Seed ist von fünf Satelliten-Seeds umgeben, die ringförmig um diesen angeordnet sind.

Abbildung 6.3: *Initial-Seeds (links) und Satelliten-Seeds (rechts).*

6.2.1 Orbit-Ermittlung

Es bietet sich an, als Region, die für die Anordnung der Satelliten-Seeds um einen Initial-Seed $\mathbf{p}_{seed}$ in Frage kommt, einen geschlossenen Dreieckszug zu wählen, der um $\mathbf{p}_{seed}$ herum verläuft. Alle Elemente des Dreieckszugs sollten zudem ungefähr gleich weit von dem Initial-Seed entfernt sein. Da auf einem solchen Dreieckszug Satelliten-Seeds angeordnet werden sollen, wird er in Analogie zur Umlaufbahn von künstlichen oder natürlichen Satelliten, die um einen Planeten kreisen, im weiteren Verlauf *Orbit* genannt. Zur Bestimmung eines geeigneten Orbits werden so genannte Ringe herangezogen. Bei einem Ring handelt es sich um eine Menge von Netzknoten, die um einen ausgewählten Netzknoten v_0 des gegebenen Oberflächennetzes angeordnet sind und von diesem Knoten gleich weit entfernt sind. Nach Gatzke et al. [GGGZ05] sind die Ringe um v_0 wie folgt definiert:

Definition 6.1 *Es sei V die Menge der Knoten eines Dreiecksnetzes, E die Menge der Kanten und T die Menge der Dreiecke. Ferner sei ein Knoten $v_0 \in V$ gegeben. Der i-te* Ring R_i um den Knoten v_0 *ist definiert als die Menge aller Knoten $v \in V$, für die gilt: Ein kürzester Kantenzug von v zu v_0 enthält genau i Kanten.*

[1] Zur besseren Veranschaulichung sind mehr Initial-Seeds gesetzt worden als eigentlich notwendig.

Dabei bestehen Kantenzüge ausschließlich aus Kanten des Dreiecksnetzes. Nach Definition 6.1 enthält ein Ring nur Netzknoten. In Abbildung 6.4 sind die ersten drei Ringe R_1 (graue Markierungen), R_2 (grün) und R_3 (blau) um den mit einer roten Markierung versehenen Knoten eingezeichnet. Die Satelliten-Seeds sollen jedoch nicht

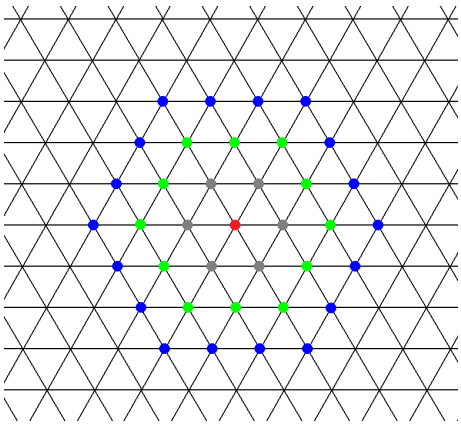


Abbildung 6.4: *Ringe R_1, R_2 und R_3 um den rot gekennzeichneten Knoten.*

um einen Netzknoten, sondern um ein Dreieck herum angeordnet werden. Um dies zu erreichen, wird Definition 6.1 wie folgt adaptiert:

Definition 6.2 *Es sei V die Menge der Knoten eines Dreiecksnetzes und T die Menge der Dreiecke. Für ein Dreieck $t_0 \in T$ sei $V_0 = \{v_0, v_1, v_2\}$ die Menge der Netzknoten, die die Eckpunkte von t_0 definieren. Der i-te* Ring R_i um t_0 *ist definiert als die Menge aller Knoten $v \in V$, für die gilt: Ein kürzester Kantenzug von v nach V_0 enthält genau i Kanten.*

Für ein reguläres Dreiecksnetz sind im linken Teilbild der Abbildung 6.5 die ersten drei Ringe um ein rot ausgefülltes Dreieck eingezeichnet. Die Farben entsprechen denen, die bereits in Abbildung 6.4 zur Veranschaulichung der Ringe um einen Knoten gewählt worden sind. Offensichtlich können aus diesen Ringen relativ intuitiv geschlossene Dreieckszüge um das rote Dreieck abgeleitet werden, die im Folgenden *Dreiecksringe* genannt werden. Drei von ihnen sind im rechten Teilbild der Abbildung 6.5 eingezeichnet. Jeder solche Dreieckszug liegt zwischen zwei benachbarten Ringen und kommt für das Platzieren der neuen Satelliten-Seeds in Frage. Formal ist ein Dreiecksring wie folgt definiert:

Definition 6.3 *Der i-te* Dreiecksring D_i *um ein Dreieck t_0 ist definiert als die Menge aller Dreiecke, die genau einen zum Ring R_i und zwei zum Ring R_{i-1} gehörenden Eckpunkte enthalten, oder bei denen dies entsprechend anders herum der Fall ist.*

Die im rechten Teilbild der Abbildung 6.5 eingezeichneten Dreieckszüge genügen dieser Definition, d.h. es handelt sich dabei um die Dreiecksringe D_1 (grau), D_2 (grün) und D_3 (blau) des roten Dreiecks. Bei regulären Dreiecksnetzen besteht ein Dreiecksring

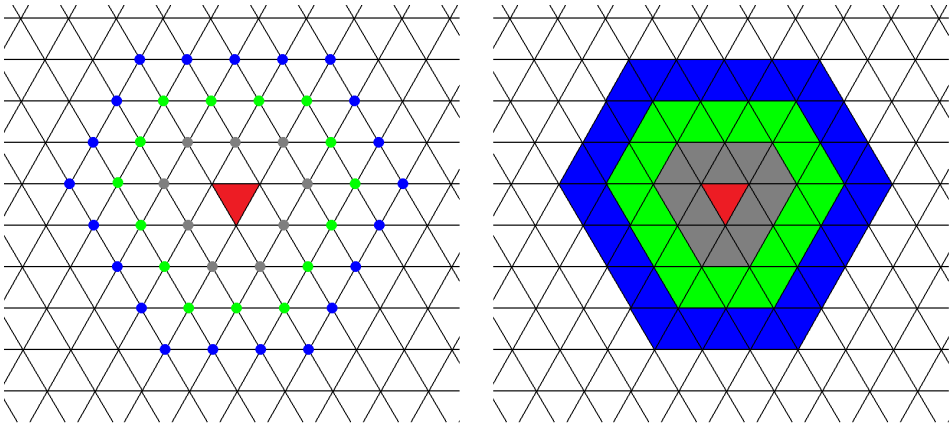


Abbildung 6.5: *Drei Ringe (links) bzw. Dreiecksringe (rechts) um das rot markierte Dreieck.*

D_i stets aus allen Dreiecken, die „innerhalb“ des Rings R_i und „außerhalb“ des Rings R_{i-1} liegen.

In Abbildung 6.5 bilden alle Punkte eines Rings R_i zusammen genau einen geschlossenen Kantenzug. Weicht hingegen ein Dreiecksnetz zumindest an einzelnen Stellen deutlich von einem regulären Netz ab, brauchen nicht alle Punkte von R_i auf dem geschlossenen Kantenzug zu liegen, der die äußere Grenze des gesuchten Dreiecksrings darstellt. Es sei $G_i^{ring} = (V_i^{ring}, E_i^{ring})$ ein Teilgraph des Graphen, der dem Dreiecksnetz zugrunde liegt. Die Menge V_i^{ring} der Knoten sei gegeben durch alle Knoten, die zu R_i gehören. Zur Menge E_i^{ring} der Kanten gehöre jede Kante des Dreiecksnetzes, die zwischen zwei Knoten des Rings R_i verläuft und die gleichzeitig einem Dreieck inzident ist, dessen dritter Knoten zum Ring R_{i-1} gehört. Im Folgenden wird ein solcher Teilgraph *Ring-Graph* genannt. Abbildung 6.6 veranschaulicht den Zusammenhang zwischen einem Ring und dem entsprechenden Ring-Graphen. Im linken Teilbild ist ein Ausschnitt eines Dreiecksnetzes dargestellt, das nicht-reguläre Strukturen aufweist. Die Ringe R_1 (grau), R_2 (grün) und R_3 (blau) um das rot hervorgehobene Dreieck sind markiert. Im rechten Teilbild ist der entsprechende Ring-Graph G_3^{ring} zu sehen. Anhand dieser Abbildung wird deutlich, dass sich die Art der Knoten eines Ring-Graphen G_i^{ring} bei nicht-regulären Dreiecksnetzen von den Knoten, die bei einem regulären Dreiecksnetz vorkommen, unterscheiden kann. Folgende zusätzliche Knoten-Arten können bei nicht-regulären Dreiecksnetzen auftreten:

- **Hängende Ring-Knoten:** Ein Knoten besitzt genau eine Kante zu einem anderen Knoten, der zur äußeren Dreiecksring-Grenze gehört. Ein solcher Fall ist durch den oberen Kreis im rechten Teilbild der Abbildung 6.6 markiert.

- **Isolierte Ring-Teilgraphen:** Mehrere Knoten des Ring-Graphen können einen unabhängigen Teilgraphen bilden. In Abbildung 6.6 ist ein solcher Teilgraph zu sehen, der aus drei Knoten besteht. In diesem Fall bildet er einen geschlossenen Kantenzug, was im Allgemeinen jedoch nicht der Fall zu sein braucht.

- **Isolierte Ring-Knoten:** Ein Knoten aus der Menge E_i^{ring} wird *isolierter Ring-Knoten* genannt, wenn er kein Endknoten einer Kante aus E_i^{ring} ist. Ein Beispiel dafür ist im rechten Teilbild der Abbildung 6.6 direkt unterhalb des hängenden Knotens zu sehen. Isolierte Ring-Knoten können als spezielle isolierte Ring-Teilgraphen angesehen werden, für die $|V_i^{ring}| = 1$ und $|E_i^{ring}| = 0$ gilt.

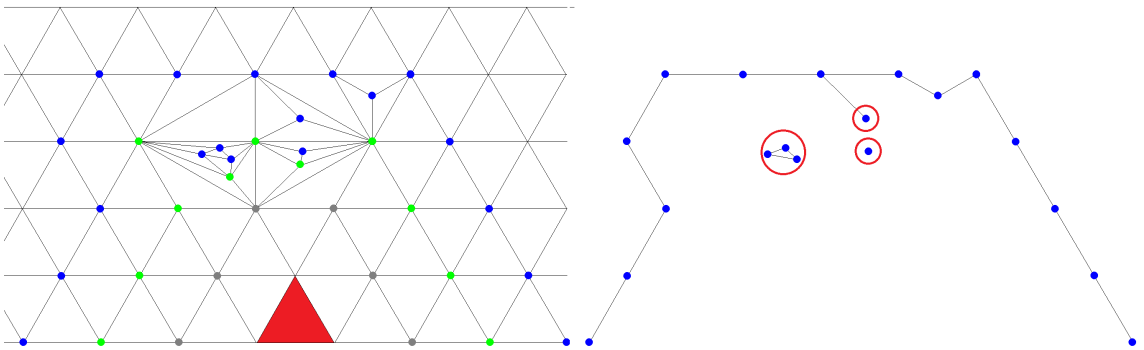


Abbildung 6.6: *Ringe um das rote Dreieck bei Vorhandensein eines nicht-regulären Netzbereichs (links) sowie der zugehörige Ring-Graph G_3^{ring} (rechts).*

Wie in Abbildung 6.6 gezeigt, können Ringe bei solchen Dreiecksnetzen, deren Struktur stark von der eines regulären Dreiecksnetzes abweicht, in unabhängige Teilgraphen zerfallen. Das Zerfallen eines Rings in einzelne Teilstücke, die nicht zusammenhängen, kann zu einem degenerierten Dreiecksring führen. Dieser Sachverhalt ist in Abbildung 6.7, der die Situation aus Abbildung 6.6 zugrunde liegt, zu sehen. Die gelben und grauen Dreiecke bilden zusammen den Dreiecksring D_3. Auch unter Berücksichtigung der Tatsache, dass es sich lediglich um einen Netzausschnitt handelt und der gelbe Dreieckszug unterhalb des roten Dreiecks weiterführt sowie geschlossen ist, lässt sich leicht zeigen, dass sich für dieses Beispiel kein Rundweg auf dem Dualgraph des Dreiecksrings konstruieren lässt, der alle zu den Dreiecksring-Dreiecken dualen Knoten enthält. Aber selbst wenn sich ein solcher Rundweg, der einem geschlossenen Dreieckszug entspricht, berechnen ließe, ist der gelb eingezeichnete Dreieckszug sicherlich für eine gleichmäßige Anordnung von Satelliten-Seeds besser geeignet.

Der degenerierte Dreiecksring aus Abbildung 6.7 stellt ein zusammenhängendes Teilnetz des gegebenen Dreiecksnetzes dar. Dies ist jedoch nicht immer der Fall. Große Dreiecksring-Indizes sowie schmale Modellteile können zu degenerierten Dreiecksringen führen, die nicht zusammenhängend sind. Ein Beispiel ist in Abbildung 6.8 anhand eines aus sehr wenigen Polygonen bestehenden Hasenmodells zu sehen. Der Dreiecksring D_2 um das blau markierte Dreieck ist gelb dargestellt. Er zerfällt in zwei getrennte Dreieckszüge. Dies hängt damit zusammen, dass das Modell durch ein äußerst grobes Dreiecksnetz gegeben ist und das Ohr somit nur durch wenige Dreiecke approximiert wird.

In den beiden geschilderten Beispielen zu degenerierten Dreiecksringen kann aus dem Dreiecksring durch Entfernen geeigneter Dreiecke ein geschlossener Dreieckszug erzeugt

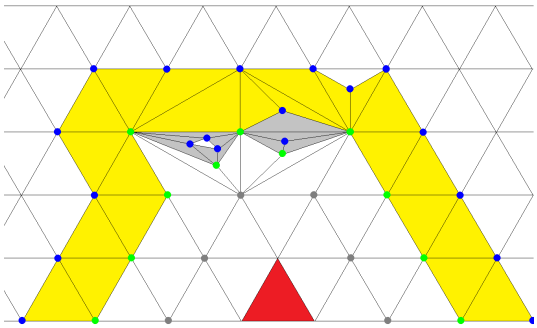


Abbildung 6.7: *Sowohl die gelben als auch die grauen Dreiecke gehören zum Dreiecksring D_3 um das rote Dreieck. Für die Anordnung von Satelliten-Seeds erscheint es jedoch sinnvoll, lediglich die gelben Dreiecke, die einen Orbit darstellen, zu berücksichtigen.*

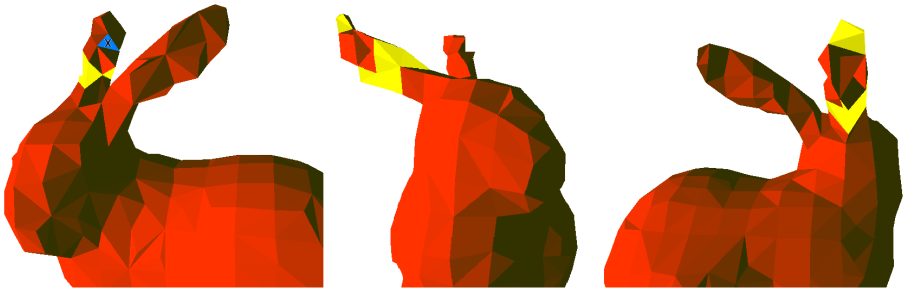


Abbildung 6.8: *Beispiel für einen Dreiecksring D_2 (gelb) um ein Seed-Dreieck (blau), der in zwei Teilstücke zerfällt.*

werden, bei dem jedes enthaltene Dreieck zu genau zwei anderen Elementen des Dreieckszugs adjazent ist. Ein solcher Dreieckszug stellt den schon angesprochenen Orbit dar und wird mit O_i bezeichnet. Dabei wird der Index i, der dem Index des zugrunde liegenden Dreiecksrings D_i entspricht, *Orbit-Index* genannt.

Um aus einem – möglicherweise degenerierten – Dreiecksring D_i einen Orbit O_i zu erzeugen, kann zunächst eine Teilmenge der Kanten des Ring-Graphen G_i^{ring} bestimmt werden, durch die der äußere Rand des Orbits definiert wird. Dazu wird ein geeigneter Kreis auf dem Dreiecksnetz benötigt, der ausschließlich Knoten und Kanten von G_i^{ring} enthält. Die Frage, welcher aller möglichen Kreise geeignet ist, kann nicht allgemeingültig beantwortet werden, da dies von der Semantik des relevanten Modell-Ausschnitts abhängt, die Semantik sich jedoch nicht ohne Weiteres automatisch ermitteln lässt. Als gute Heuristik hat sich allerdings herausgestellt, die Größe des Kreises, also die Länge des entsprechenden Kantenzuges, als Qualitätskriterium heranzuziehen: Je länger ein Kantenzug ist, umso besser ist er als Grenze für den Orbit geeignet. Als Wahl für die Länge scheinen sich prinzipiell die absolute Länge der entsprechenden Kontur auf der Dreiecksfläche oder die Anzahl der Knoten bzw. Kanten des Kantenzuges zu eignen. Es seien $\hat{E}_i^{ring} \subseteq E_i^{ring}$ und $\hat{V}_i^{ring} \subseteq V_i^{ring}$ die Mengen der Kanten und Knoten eines längsten geschlossenen Kantenzuges, für den zusätzlich gilt, dass für jedes $e \in \hat{E}_i^{ring}$ ein zu e inzidentes Dreieck existiert, welches einen zum Ring R_{i-1} gehörenden Eckpunkt besitzt. Dann enthält der Orbit genau alle Dreiecke, die zum Dreiecksring D_i gehören und die zu mindestens einem Knoten aus $\hat{V}_i^{ring}$ inzident sind.

6.2.2 Anordnung der Satelliten-Seeds auf dem Orbit

Die Satelliten-Seeds eines Initial-Seeds $\mathbf{p}_{seed}$ sind möglichst gleichmäßig auf einem Orbit um $\mathbf{p}_{seed}$ herum anzuordnen. Der zu wählende Orbit-Index hängt dabei von der Anzahl der Satelliten-Seeds ab. Weiter unten wird darauf näher eingegangen. Anzumerken bleibt, dass in Spezialfällen die Konstruktion eines geschlossenen Orbits zu einem gewünschten Orbit-Index nicht möglich ist. Solche Fälle sind jedoch nur äußerst selten zu erwarten. Nicht zuletzt deswegen wird, sobald eine solche Situation vorliegt, der entsprechende Initial-Seed lediglich übernommen, ohne dass weitere Satelliten-Seeds hinzugefügt werden. Dies scheint auch insofern legitim zu sein, als dass das Oberflächennetz in der betrachteten Region offensichtlich eher ungewöhnliche Strukturen aufweist. Eine detaillierte Analyse des Auftretens solcher Situationen erfolgt in Abschnitt 6.4.

Es sei $n_{sat} \in \mathbb{N}_0$ die Anzahl der Satelliten-Seeds, die für jeden einzelnen Initial-Seed zu ermitteln sind. Diese Satelliten-Seeds sollen derart angeordnet werden, dass sie recht nah am zugehörigen Initial-Seed liegen, von diesem alle ungefähr gleich weit entfernt sind und auch einen gewissen Mindestabstand voneinander haben. Der Forderung, dass die Entfernung zum Initial-Seed ungefähr gleich sein soll, kann entsprochen werden, indem die zusammengehörenden Satelliten-Seeds auf demselben Orbit angeordnet werden. Welcher Orbit dafür zu wählen ist, muss in Abhängigkeit von n_{sat} entschieden werden, damit die anderen genannten Forderungen eingehalten werden können.

Die Anzahl der Satelliten-Seeds sollte nicht zu groß sein, da die für die Patch-Ermittlung benötigte Rechenzeit mit steigender Anzahl von Seed-Punkten zunimmt. Dies ist insbesondere bei der in Kapitel 7 beschrieben Optimierung von besonderer Bedeutung. Darüber hinaus sind auch gezielte Eingriffe bei zu vielen Seed-Punkten nicht mehr gut möglich. In Abhängigkeit von der Anzahl n_{sat} der zu platzierenden Satelliten-Seeds ist ein geeigneter Orbit-Index zu wählen. Ein solcher Orbit muss zumindest groß genug sein, um alle n_{sat} Satelliten-Seeds aufnehmen zu können. Liegen Seed-Punkte zu nahe aneinander, ist die Wahrscheinlichkeit sehr groß, dass einige von den anderen dominiert werden, also bei der Berechnung des Patches keine Rolle spielen. Aus diesem Grund sollten keine zwei aufeinanderfolgende Elemente des Orbits als Seed-Dreiecke gewählt werden. Andererseits darf der Abstand auch nicht so groß werden, dass sich durch das Satelliten-Seeding die Gestalt der Patches grundlegend ändert.

Da Dreiecksnetze prinzipiell eine beliebige Struktur haben können, kann kein direkter Zusammenhang zwischen Orbit-Index und Anzahl der entsprechenden Orbit-Elemente quantifiziert werden. Um dennoch einen Orbit-Index abzuschätzen, der für viele Modelle geeignet ist, wird das reguläre Dreiecksnetz aus Abbildung 6.5 betrachtet. Dies erscheint gerechtfertigt, da die Struktur zahlreicher in dieser Arbeit untersuchter dreidimensionaler Modelle nicht allzu sehr von der eines regulären Netzes abweicht. Reguläre Dreiecksnetze haben die Eigenschaft, dass Orbit und Dreiecksring stets identisch sind. Der i-te Orbit um das rote Dreieck in Abbildung 6.5 enthält genau $12 \cdot i$ Dreiecke. In dieser Arbeit wird als Heuristik verwendet, dass bei regulären Dreiecksnetzen zwischen den Seed-Dreiecken auf dem Orbit möglichst vier, jedoch nicht weniger als drei „normale“ Dreiecke liegen. Daraus ergibt sich

$$\begin{aligned} 4 \cdot n_{sat} &\leq 12 \cdot i_{ideal} \\ \Rightarrow i_{ideal} &\geq \frac{n_{sat}}{3}. \end{aligned} \tag{6.1}$$

Da der Orbit aber auch nicht viel mehr Elemente als notwendig enthalten soll, ist demnach der optimale Orbit-Index durch $i_{ideal} = \lceil \frac{n_{sat}}{3} \rceil$ gegeben. Ein beliebiges Dreieck des Orbits $O_{i_{ideal}}$ wird als erster Satelliten-Seed gewählt. Die weiteren $(n_{sat} - 1)$ Satelliten-Seeds werden möglichst gleichmäßig über $O_{i_{ideal}}$ verteilt. Dabei wird der dreieckbasierte Abstand zugrunde gelegt, da die Anzahl der Elemente des Orbits bereits grob abgeschätzt worden ist, nicht jedoch deren Größe oder genaue Lage. Bei anderen Abstandsmassen wie beispielsweise dem geodätischen oder dem winkelbasierten wäre die Wahrscheinlichkeit ungleich größer, dass zwei Satelliten-Seeds auf benachbarten Dreiecken platziert werden.

6.2.3 Ausnahmefälle

Abhängig von der konkreten Netzstruktur kann es zu Situationen kommen, in denen die Bestimmung von Satelliten-Seeds nicht sinnvoll erscheint. Ein erster solcher Sonderfall ist bereits weiter oben angesprochen worden: Wenn für einen Initial-Seed kein Orbit in Form eines geschlossenen Dreieckszugs existiert, werden für diesen keine Satelliten-Seeds ermittelt. Daneben gibt es mit *überlappenden Orbits* noch einen zweiten Ausnahmefall, bei dem ebenfalls für die betreffenden Initial-Seeds kein Satelliten-Seeding

durchgeführt wird: Wenn Seed-Punkte, für die ein Satelliten-Seeding durchgeführt werden soll, zu nahe aneinander liegen, können deren Orbits dieselben Elemente enthalten. Würden diese Orbits dennoch zum Platzieren der Satelliten-Seeds herangezogen, könnten Patches entstehen, die in unzusammenhängende Teile zerfallen. Dieses gilt insbesondere für die anschließend anwendbare evolutionäre Patch-Optimierung, die in Kapitel 7 beschrieben wird. Da auf überlappenden Orbits keine weiteren Seed-Punkte platziert werden, wird somit automatisch für Seed-Linien die Ermittlung von Satelliten-Seeds unterbunden.

Ein Satelliten-Seeding wird für Initial-Seeds durchgeführt. Wie schon Abbildung 4.2 des Überblick-Kapitels verdeutlicht, kann die Ermittlung von Satelliten-Seeds beim EvOpSeg-Verfahren bedingt durch manuelle Eingriffe der Benutzer durchaus häufiger als nur einmal erfolgen. Das ist dann der Fall, wenn nach einem bereits durchgeführten Satelliten-Seeding weitere Initial-Seeds von Hand platziert werden, für die ebenfalls Satelliten-Seeds zu bestimmen sind. Auch für vorhandene Satelliten-Seeds können unter Umständen weitere Satelliten-Seeds erzeugt werden; die vorhandenen stellen dann in gewissem Sinne selbst eine Art Initial-Seeds dar. Damit eine Erzeugung von Satelliten-Seeds für existierende Satelliten-Seeds möglich ist, müssen allerdings vorher Seed-Punkte verschoben worden sein. Ansonsten würden die Orbits einander durchdringen, so dass keine neuen Satelliten-Seeds angeordnet werden. Diese Aussage gilt zumindest solange, wie der Wert von n_{sat} und damit der Orbit-Index nicht signifikant verringert wird.

6.3 Berechnung der Sub-Patches

Ebenso wie die Initial-Seeds verfügen auch die einzelnen Satelliten-Seeds über individuelle Gewichtswerte. Da ein durch mehrere Sub-Patches gegebenes Patch weitgehend mit dem Patch übereinstimmen soll, das vor der Anwendung des Satelliten-Seeding durch den entsprechenden Initial-Seed vorgelegen hat, bietet es sich an, die Gewichtswerte der Initial-Seeds für die Satelliten-Seeds zu übernehmen. Dieser Zusammenhang lässt sich wie folgt formalisieren: Es sei S^{init} die Menge aller Initial-Seeds und $\mathbf{p}_i$ der i-te Initial-Seed-Punkt[2]. Ferner sei S_i^{sat} die Menge aller zum Initial-Seed $\mathbf{p}_i$ gehörenden Satelliten-Seeds und $\mathbf{p}_{i,j}^{sat}$ das j-te Element aus S_i^{sat}. Die Gewichtswerte der Satelliten-Seeds werden so initialisiert, dass für alle $i \in \{1, \ldots, |S^{init}|\}$ und für alle $j \in \{1, \ldots, |S_i^{sat}|\}$ gilt: $\gamma_{i,j} = \gamma_i$, wobei γ_i den Gewichtswert von $\mathbf{p}_i$ und $\gamma_{i,j}$ den des Satelliten-Seeds $\mathbf{p}_{i,j}^{sat}$ bezeichnen. Die Übernahme der Gewichtswerte von den Initial-Seeds bewirkt zusammen mit der lokal begrenzten Anordnung der Satelliten-Seeds (vgl. Abschnitt 6.2), dass die Ränder der Patches durch das Satelliten-Seeding zunächst kaum beeinflusst werden. Eine bewusste Beeinflussung ist jedoch im Anschluss durch manuelle Eingriffe oder automatische Optimierungen der Seed-Gewichte sowie der Seed-Positionen möglich. Auf die automatische Optimierung wird in Kapitel 7 ausführlich eingegangen.

[2] An dieser Stelle wird davon ausgegangen, dass die Elemente der Menge S^{init} in Form einer indizierten Liste vorliegen. Gleiches gilt auch für die nachfolgend erwähnten Mengen S_i^{sat}.

Bereits bei der Initialsegmentierung konnte ein Patch durchaus mehrere Seed-Punkte in Form allgemeiner Seed-Cluster enthalten. Satelliten-Seeds stellen eine weitere, spezielle Art von Seed-Clustern dar. Da der in Kapitel 5 angegebene Algorithmus zur Patch-Berechnung (vgl. Listing 5.1) auch für Seed-Cluster ausgelegt ist, kann er ohne größere Änderungen ebenfalls bei Satelliten-Seeds angewendet werden. Es wird lediglich für jedes Dreieck zusätzlich nachgehalten, zu welchem Sub-Patch es gehört.

6.4 Ergebnisse

Notwendige Bedingung für die Ermittlung des Orbits zu einem gegebenen Orbit-Index ist die Existenz eines entsprechenden Kreises auf dem Ring-Graphen um den Initial-Seed $\mathbf{p}_{seed}$. Wie in Abschnitt 6.2 erwähnt, kann es Situationen geben, in denen kein Orbit berechnet werden kann. Der Orbit-Konstruktion lag jedoch die Annahme zugrunde, dass solche Situationen nur selten auftreten. Im weiteren Verlauf wird genauer untersucht, wie häufig damit zu rechnen ist. Anschließend werden die Patches der Initialsegmentierung mit denen verglichen, die sich bei Verwendung der Satelliten-Seeds ergeben. Im Idealfall sollten sie weitestgehend übereinstimmen. Abschließend wird gezeigt, dass sowohl ein Verschieben der Seeds als auch eine Gewichtsanpassung bei Verwendung der Satelliten-Seeds zu besseren Ergebnissen führen kann, als wenn lediglich Initial-Seeds vorhanden sind, und sich das Satelliten-Seeding somit rentiert. Dabei ist in den Beispielen stets für die Anzahl der Satelliten-Seeds $n_{sat} = 5$ gewählt worden.

Orbit-Berechnung

Die Ermittlung eines Orbits erfolgt mit Hilfe geeigneter Kreise auf Ring-Graphen. Wenn es für einen Seed-Punkt keinen solchen Kreis gibt, existiert demnach auch kein Orbit. In einer umfangreichen Versuchsreihe ist untersucht worden, wie groß die Wahrscheinlichkeit dafür ist. Anhand von rund 400 Modellen, die aus vollkommen unterschiedlichen Domänen stammen[3], sind mehr als 7600 Ringe R_2 bestimmt worden. Die Initial-Seeds sind dabei nach dem in Abschnitt 5.1.2 beschriebenen automatischen Seeding ermittelt worden. In lediglich einem einzigen Fall gab es einen Initial-Seed, für den kein Kreis auf dem Ring-Graph von R_2 (und damit dann auch kein Orbit) bestimmt werden konnte. Der Grund dafür liegt in der Tatsache, dass das zugehörige Dreiecksnetz nicht wasserdicht ist, also dass nicht jedes Dreieck t drei Nachbardreiecke besitzt, die mit t eine Kante gemeinsam haben. Dadurch resultieren „Löcher“ in dem Modell; ein solches befindet sich beispielsweise im linken Teilbild von Abbildung 6.9 zwischen den Vorderläufen des Hasen. Der Initial-Seed, für den kein Kreis ermittelt werden konnte, liegt direkt an einem Loch, so dass dieses Loch einen Kreis auf dem Ring-Graph unterbindet.

In allen anderen Fällen war die Ermittlung des Kreises hingegen möglich. Dabei lagen alle weiteren getesteten Modelle als wasserdichte Dreiecksnetze vor. Dies legt die

[3] 380 dieser Modelle stammen aus dem so genannten Princeton-Benchmark [CGF09].

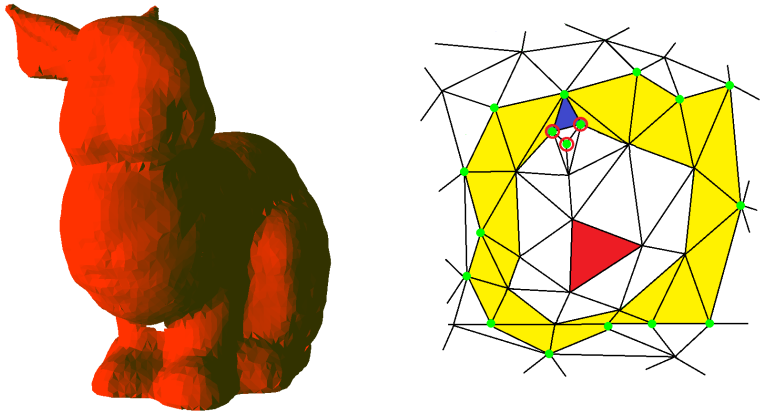


Abbildung 6.9: *Löcher in Dreiecksnetzen (links, zwischen den Vorderläufen) sowie stark irreguläre Strukturen (rechts) können bei der Orbit-Bestimmung problematisch sein.*

Vermutung nahe, dass bei wasserdichten Netzen, die nicht stark degeneriert sind, der gewünschte Kreis stets existiert.

Ohne einen Kreis auf dem Ring-Graph ist auch kein Orbit definiert. Andererseits sagt die Existenz eines solchen Kreises noch nichts über das Vorhandensein eines Orbits aus. In insgesamt fünf der mehr als 7600 Fälle war keine Orbit-Berechnung möglich. Das Problem ist stets bei Modellen aufgetreten, die Dreiecke mit stark unterschiedlichen Größen besitzen. Als exemplarische Veranschaulichung dient das rechte Teilbild aus Abbildung 6.9. Alle zum Ring R_2 um das rote Dreieck gehörenden Knoten sind grün markiert. Die drei rot umrandeten Knoten gehören jedoch nicht zum entsprechenden Kreis, da ansonsten der obere Knoten des blauen Dreiecks doppelt im Kreis enthalten wäre. Alle Dreiecke, die zum Orbit mit dem Orbit-Index 2 gehören, müssen mindestens einen Knoten des Rings R_1 enthalten. In Abbildung 6.9 sind sie gelb markiert. Das blaue Dreieck enthält ausschließlich Knoten des Rings R_2. Folglich gehört es nicht zum Orbit, so dass für diesen Fall kein geschlossener Orbit O_2 existiert.

Insgesamt sind trotz zahlreicher und vollkommen unterschiedlicher Modelle in lediglich 0.065% der Fälle Situationen aufgetreten, in denen kein Orbit ermittelt werden konnte. Die Wahrscheinlichkeit dafür ist also äußerst gering. Dieses sowie die Tatsache, dass in erster Linie nur degenerierte und nicht-wasserdichte Netze dazu tendieren, rechtfertigen das Vorgehen, in entsprechenden Fällen lediglich den Initial-Seed zu übernehmen.

Gestalt der Patches

Initial-Seeds ermöglichen es, relativ schnell eine erste Zerlegung des Modells in Patches durchzuführen, die bei guter Wahl der Seed-Positionen tendenziell bedeutungsvollen Komponenten entsprechen. Dabei können die Anwendenden die Gestalt der Patches bereits grob durch manuelle Eingriffe beeinflussen. Präzisere Eingriffsmöglichkeiten

werden durch das anschließende Hinzufügen von Satelliten-Seeds geschaffen. Damit die Satelliten-Seeds vorherige Anpassungen nicht konterkarieren, dürfen sich die Patches durch das Satelliten-Seeding nicht zu sehr verändern. Im Rahmen der Bearbeitung des Themas ist der Einfluss des Hinzufügens von Satelliten-Seeds auf die Patches untersucht worden. In den meisten Fällen hat sich keine oder keine bedeutende Veränderung ergeben. Ein Beispiel dafür ist in Abbildung 6.10 angegeben. Im linken Teilbild ist die Initial-Segmentierung eines Krugs dargestellt, die bereits als Beispiel in Abbildung 5.13 zu sehen war. Das Hinzufügen von Satelliten-Seeds führt zu keiner erkennbaren Veränderung der Patches. Diese Tatsache ist hier als positiv anzusehen, da dann insbesondere auch keine schlechteren Patches entstehen. Eine Verbesserung der Patch-Grenzen lässt sich anschließend in vielen Fällen mit dem in Kapitel 7 beschriebenen Optimierungsansatz erzielen.

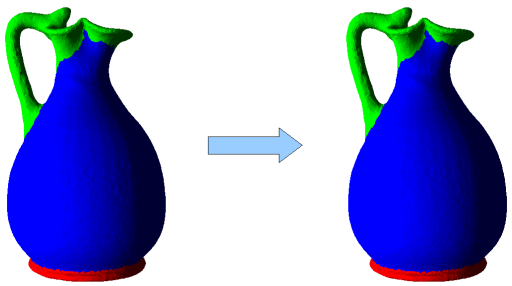


Abbildung 6.10: *Die Struktur der durch Initial-Seeds definierten Patches (linkes Teilbild) bleibt in diesem Fall auch bei Verwendung von Satelliten-Seeds erhalten (rechtes Teilbild).*

Ob ein Hinzufügen von Satelliten-Seeds zu signifikanten Veränderungen der Patches führt, hängt stark von der Qualität der Initial-Seed-Positionen ab. Sobald sich Initial-Seeds in direkter Nähe von konkaven Komponentenrändern befinden, kann das Satelliten-Seeding dazu führen, dass Patches über diese Grenze hinauslaufen. Dies wird anhand von Abbildung 6.11 veranschaulicht. Jedes Patch des linken Teilbilds enthält genau einen zentral liegenden Seed-Punkt. Das Satelliten-Seeding führt nur zu vernachlässigbar kleinen Veränderungen der Patch-Ränder (zweites Teilbild). Wird nun jedoch der das pinkfarbene Patch definierende Initial-Seed in die Nähe des Randes zum grünen Sockel angeordnet (drittes Teilbild), führt das Satelliten-Seeding dazu, dass nun nicht mehr alle Satelliten-Seeds auf dem Fuß des Kruges liegen (viertes Teilbild). Dadurch verschlechtert sich die Grenze zwischen den beiden betroffenen Patches. Zur Verringerung solcher negativer Einflüsse bieten sich unter anderem folgende Lösungsmöglichkeiten an:

a. Wenn ein Initial-Seed in der Nähe eines Patch-Randes liegt bzw. insbesondere dann, wenn der Orbit durch mehrere Patches verläuft, werden für diesen Initial-Seed keine Satelliten-Seeds erzeugt.

b. Wenn ein Orbit durch mehrere Patches verläuft, werden Satelliten-Seeds nur auf solchen Orbit-Bereichen angeordnet, die innerhalb des Patches liegen, zu dem der entsprechende Initial-Seed gehört.

Im Gegensatz zu der ersten Lösungsmöglichkeit bietet die zweite mehr Freiheitsgrade für eine nachfolgende Patch-Optimierung. In der vorliegenden Dissertation haben jedoch diese beiden Lösungsmöglichkeiten keine Anwendung gefunden, da die Ränder der Patches der Initial-Seeds durchaus weit von den idealen Komponentenrändern entfernt liegen können. Dies ist häufig dann der Fall, wenn ein Initial-Seed innerhalb einer konkaven Region, also einer „Vertiefung“ der Komponente, liegt und das Patch damit kleiner ausfällt als erwartet. In solchen Fällen ist die Veränderung der Patch-Struktur durch die Verwendung von Satelliten-Seeds sogar von Vorteil.

Insgesamt gab es bei der durchgeführten Testreihe deutlich mehr Situationen, in denen Patches nach Anwendung des Satelliten-Seeding zumindest annähernd mit den vorherigen Patches übereinstimmten, als dass dies nicht der Fall war. Insbesondere bei manuell gesetzten Initial-Seeds scheint es recht unproblematisch zu sein, da diese üblicherweise zentral auf den Komponenten angeordnet werden.

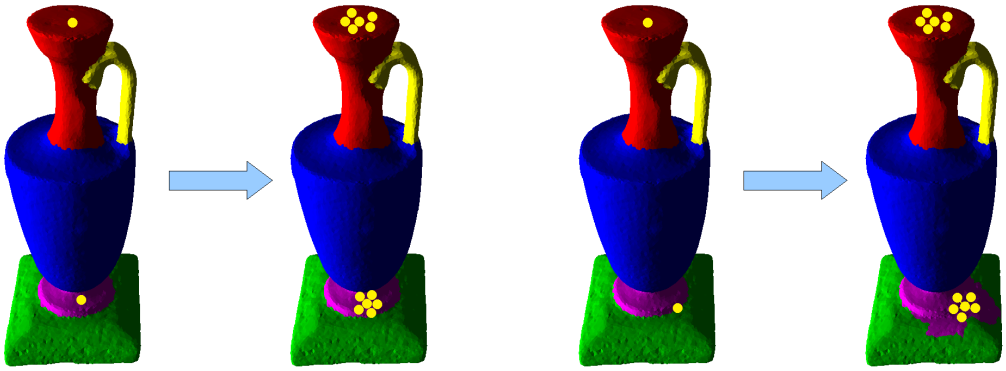


Abbildung 6.11: *Vergleich der Patches mit Initial-Seeds (Teilbilder 1 und 3) und Satelliten-Seeds (Teilbilder 2 und 4).*

Einfluss von Gewichtsadaptionen bei Verwendung von Satelliten-Seeds

Das Satelliten-Seeding wurde eingeführt, um eine feinere bzw. genauere Anpassung der Patches zu ermöglichen. Solche Anpassungen können durch manuelle Eingriffe, insbesondere durch Verändern der Seed-Gewichte sowie der Seed-Positionen, vorgenommen werden. Eine automatische Veränderung dieser Parameter ist ebenfalls möglich; Kapitel 7 befasst sich ausführlich mit diesem Aspekt. Zur Demonstration des Potentials, das Satelliten-Seeds bieten, werden die beiden Initialsegmentierungen aus Abbildung 6.12 als Ausgangssituation betrachtet. Jedes dargestellte Patch der Ente besitzt genau einen Seed-Punkt. Beim zweiten Modell sind hingegen Seed-Cluster verwendet worden: Das Hemd besitzt zwei Seed-Punkte (einen auf der Vorder- und einen auf der Rückseite),

die Hose wird von insgesamt drei Seed-Punkten definiert. Beide Segmentierungen weisen Schwachstellen auf. So ragt der segmentierte Schnabel der Ente auf einer Seite deutlich in den Kopf hinein. Alle anderen Komponenten-Grenzen sind jedoch recht gut getroffen. Die Segmentierung des Mann-Modells aus Abbildung 6.12 besitzt gleich zwei schlechte Patch-Grenzen: Weder der Übergang von der Hose zum Oberkörper noch der vom Oberkörper zu dem aus Sicht des Betrachters rechts liegenden Arm entsprechen den Erwartungen. Die Patches, die durch ein Satelliten-Seeding entstehen, sind zu denen aus Abbildung 6.12 nahezu identisch.

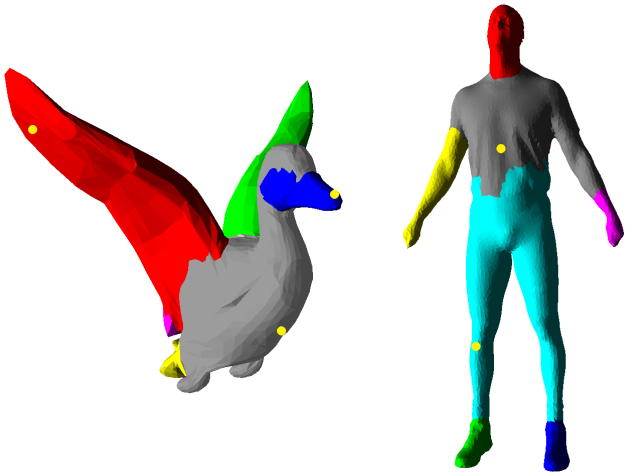


Abbildung 6.12: *Anhand dieser beiden Initial-Segmentierungen wird gezeigt, dass Satelliten-Seeds das Erlangen von korrekten Segmentierungen durch Änderungen der Seed-Gewichte (s. Abbildung 6.13) oder der Seed-Positionen (s. Abbildung 6.14) begünstigen.*

Die linke Spalte in Abbildung 6.13 zeigt die besten aller erzielten Ergebnisse, die sich durch manuelle Anpassung der Gewichtswerte der Initial-Seeds ergeben haben. Dabei blieben die Positionen der Seed-Punkte unverändert. Das Patch, das den Schnabel der Ente darstellt, ragt nun nicht mehr zu weit in den Kopfbereich hinein. Dafür ist aber die obere Schnabel-Grenze nicht allzu gut approximiert. Ganz anders sieht das bei Verwendung von Satelliten-Seeds aus (rechte Spalte): Der Schnabel wird nach manueller Anpassung einzelner Seed-Gewichte perfekt segmentiert.

Ohne Verwendung des Satelliten-Seeding können die Patches des Mann-Modells durch alleiniges Anpassen der Seed-Gewichte nur bedingt verbessert werden. Das Ergebnis ist links unten in Abbildung 6.13 zu sehen. Die verbesserte Segmentierung des Arms ist erreicht worden, indem das Seed-Gewicht des Arms um 40 Prozent vergrößert wurde. Trotz bestmöglicher Abstimmung der Seed-Gewichte für den Oberkörper sowie für die Hose wird kein guter Übergang zwischen diesen beiden Modellbereichen erzielt, da das blaue Patch auf der rechten Seite signifikant in den Bereich des Oberkörpers hineinragt. Ganz anders sieht das bei Verwendung von Satelliten-Seeds aus, wie im Teilbild rechts unten zu sehen ist. Im Gegensatz zu der Variante ohne Satelliten-Seeds sind dabei ausschließlich die Gewichte einiger der Satelliten-Seeds angepasst worden,

die das graue Patch definieren. In Richtung des Arms ist das Gewicht verringert worden, in Richtung der Hose vergrößert. Das Ergebnis entspricht prinzipiell den Erwartungen. Die Grenzlinie könnte zwar etwas „glatter“ sein, dies lässt sich jedoch sicherlich durch Anwendung eines Boundary-Smoothing-Verfahrens (vgl. [KT09, YSOM10]) oder auch durch die in Kapitel 7 beschriebene evolutionäre Optimierung beheben.

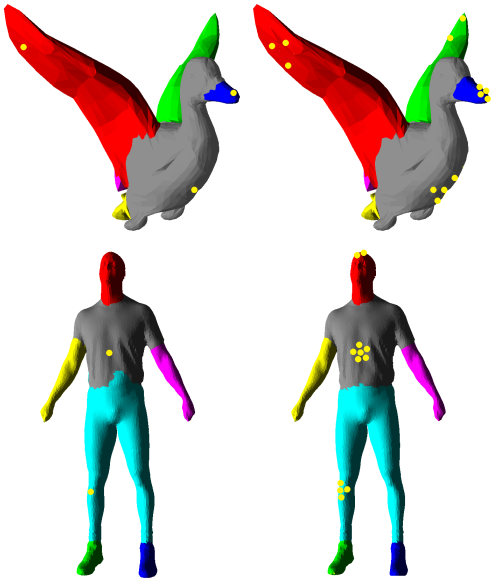


Abbildung 6.13: *Während eine Veränderung des Seed-Gewichts bei Initial-Seeds nicht zum Erfolg führt (linke Spalte), resultieren aus der Anpassung von Seed-Gewichten bei Verwendung von Satelliten-Seeds Patches, die bedeutungsvollen Komponenten entsprechen (rechte Spalte).*

Einfluss von Seed-Verschiebungen bei Verwendung von Satelliten-Seeds

Anhand der in Abbildung 6.12 gezeigten Ausgangssituationen lässt sich auch der Nutzen des Satelliten-Seeding bei der Patch-Anpassung durch Verschieben einzelner Seed-Punkte veranschaulichen. In der linken und mittleren Spalte von Abbildung 6.14 sind Ergebnisse dargestellt, die sich ohne das Vorhandensein von Satelliten-Seeds durch geeignete Verschiebung der Initial-Seeds ergeben. Bessere Ergebnisse als die dargestellten wurden nicht erzielt. Bei der Ente bewirkt erst eine relativ starke Verschiebung des Initial-Seeds, der das „Körper-Patch“ definiert, eine entscheidende Veränderung der Patches; wird der Seed nicht über den Hals hinaus verschoben, scheint keine signifikante Verbesserung erreichbar zu sein (Teilbild oben links). Andernfalls wird zwar der Schnabel korrekt segmentiert, allerdings fällt dabei der Rumpf mit dem roten Flügel zu einem einzigen Patch zusammen (oben Mitte). Bei Verwendung von Satelliten-Seeds kann hingegen durch Verschiebung einzelner Seed-Punkte bewirkt werden, dass sowohl das den Schnabel approximierende Patch als auch alle übrigen Patches den erwarteten Komponenten entsprechen.

Ähnliche Ergebnisse ergeben sich auch für das zweite Beispiel, wie anhand der unteren Zeile von Abbildung 6.14 deutlich wird. Ohne Satelliten-Seeding kann eine Verschiebung der Initial-Seeds nicht die gesamte Grenze zwischen der Hose und dem Oberkörper positiv beeinflussen. Je nach konkreter Anordnung der Seeds läuft das hellblaue Patch links oder rechts in den Bereich des Oberkörpers hinein (Teilbilder unten links bzw. Mitte). Eine leichte Verschiebung des Initial-Seeds, welcher in der Ausgangssituation das rosafarbene Patch definiert, führt dazu, dass nun der gesamte Arm einschließlich der Schulter ein Patch ergibt. Dies entspricht ebenfalls nicht unbedingt den Erwartungen. Wie schon bei dem Entenmodell erweisen sich auch hier die Satelliten-Seeds als äußerst hilfreich. Im rechten Teilbild sind die Patches eingezeichnet, die sich durch Verschiebung weniger Satelliten-Seeds ergeben. Sämtliche Grenzen zwischen den Patches folgen den intuitiv erkennbaren Komponentengrenzen. Offensichtlich hilft das Satelliten-Seeding nicht nur bei der Patch-Optimierung durch Anpassung der Gewichtswerte, sondern auch bei einer Optimierung durch Ändern der Seed-Positionen.

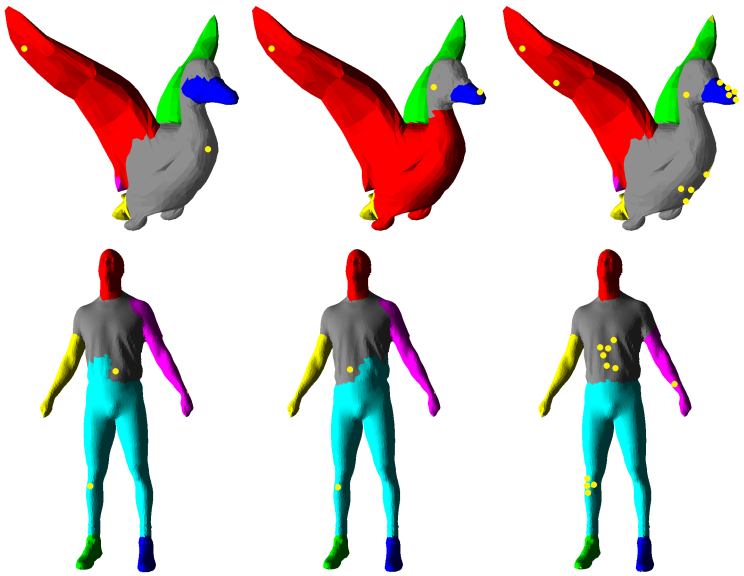


Abbildung 6.14: *Während eine Verschiebung der Initial-Seeds nicht zum Erfolg führt (linke und mittlere Spalte), resultieren aus der Verschiebung von Seed-Punkten bei Verwendung von Satelliten-Seeds Patches, die bedeutungsvollen Komponenten entsprechen (rechte Spalte).*

6.5 Fazit

Es wurde gezeigt, dass Initial-Seeds alleine oft nicht ausreichen, um ein gutes Segmentierungsergebnis zu erhalten. Wird versucht, durch manuelle Eingriffe einen Randabschnitt eines Patches zu verbessern, kann dies durchaus zu einer Verschlechterung an anderen Abschnitten des Patch-Randes führen. Satelliten-Seeds wirken dem entgegen

und haben ihr Potential sowohl bei der Gewichtsadaption als auch beim Verschieben von Seeds gezeigt. Die zu einem Initial-Seed gehörenden Satelliten-Seeds werden auf einem Orbit um diesen herum angeordnet. Abhängig von der Struktur des Dreiecksnetzes braucht nicht unbedingt zu jedem Seed ein gewünschter Orbit zu existieren. So können beispielsweise Löcher in den Modellen dazu führen, dass kein Orbit vorliegt. Durch einen umfangreichen Test wurde allerdings gezeigt, dass es sich um Ausnahmesituationen handelt, die deutlich seltener als in einem Promille der Fälle auftreten.

Die Patches, die sich durch das Satelliten-Seeding ergeben, sind den ursprünglichen häufig sehr ähnlich. Als etwas problematisch erweist sich das Satelliten-Seeding in Situationen, in denen bereits die Initial-Seeds eher schlecht auf dem Modell positioniert sind: Liegt ein Initial-Seed direkt am Rand einer logischen Komponente, kann das Hinzufügen von Satelliten-Seeds dazu führen, dass sich das Patch dadurch zu weit in die danebenliegende Komponente hinein erstreckt. Dieses Problem ist dann allerdings eher dem Initial-Seeding zuzuschreiben als dem Satelliten-Seeding-Ansatz. Insbesondere bei manuell gesetzten Initial-Seeds sollte dieses Problem nicht auftreten, sofern die Seeds möglichst zentral auf den Komponenten platziert werden.

Kapitel 7

Evolutionäre Patch-Optimierung

Der in den vorangegangenen Kapiteln vorgestellte Algorithmus zur Patch-Berechnung ist nicht speziell für bestimmte Modellklassen entwickelt bzw. an die spezifischen Eigenschaften einzelner Klassen angepasst worden. Vielmehr soll er einen möglichst universellen Charakter haben. Dadurch können die erzeugten Patches allerdings nicht in allen Fällen optimal sein, ganz unabhängig davon, ob nur Initial-Seeds oder zusätzlich auch Satelliten-Seeds verwendet worden sind. Eine Verbesserung der Patch-Ränder kann beispielsweise durch manuelle Anpassungen erreicht werden (vgl. Abschnitt 5.3), was sich bei der Anwendung des Satelliten-Seeding im Allgemeinen leichter bzw. zielgerichteter steuern lässt, als wenn ausschließlich Initial-Seeds vorliegen.

In diesem Kapitel wird mit der evolutionären Patch-Optimierung ein Ansatz zur automatischen Verbesserung der Patches vorgestellt. Die Optimierung erfolgt mit Hilfe *Evolutionärer Algorithmen* [BS93], die recht flexibel sind und sich bereits in zahlreichen unterschiedlichen Domänen bewährt haben. Als Beispiele seien hier das Traveling Salesperson Problem (TSP) [Rud91, Pri04], die Entwicklung optimierter Laufmuster von zwei- bzw. vierbeinigen Robotern [Heb09], die Trajektorienplanung von Manipulatorarmsystemen [Ort02], die Analyse von intravaskulären Ultarschallbildern [LEB+06] oder die Optimierung von Musikstücken bzw. Melodien [Hoc06] genannt. Auch für die Optimierung der Patches lassen sich Evolutionäre Algorithmen gut nutzen. Als Vorteil der Evolutionären Algorithmen gegenüber einigen anderen Optimierungsansätzen zur Verbesserung von Patch-Rändern ist zu nennen, dass mit Evolutionären Algorithmen nach einem globalen Optimum gesucht wird, welches mitunter „jenseits“ eines zwischendurch gefundenen lokalen Optimums liegen kann.

Im ersten Unterabschnitt wird zunächst auf Evolutionäre Algorithmen im Allgemeinen sowie auf Evolutionsstrategien als eine konkrete Ausprägung eingegangen. Im Anschluss daran befasst sich Abschnitt 7.2 mit der Anwendung einer Evolutionsstrategie zur Optimierung von Patches. Abschnitt 7.3 führt eine zweistufige Patch-Optimierung ein. Parallelisierungsmöglichkeiten werden in Abschnitt 7.4 diskutiert, bevor in Abschnitt 7.5 Optimierungsresultate vorgestellt werden.

7.1 Evolutionäre Algorithmen

Es gibt Optimierungsprobleme, deren Suchräume zu komplex für eine erschöpfende Suche sind und über deren Struktur zu wenige Kenntnisse vorhanden sind, als dass eine optimale Lösung analytisch bestimmt werden könnte. Für solchen Problemstellungen bieten sich randomisierte Suchheuristiken wie beispielsweise *Evolutionäre Algorithmen* an. Den Evolutionären Algorithmen liegt als Vorbild die biologische Evolution zugrunde (vgl. bspw. [BS93, Bäc96]). Eine mögliche Lösung des Optimierungsproblems wird dabei als *Individuum* bezeichnet. In der Regel werden pro Generation mehrere Individuen betrachtet, die zusammen eine *Population* bilden; es existieren aber auch Varianten, bei denen pro Generation lediglich ein Individuum berücksichtigt wird, d.h. jede Population besteht aus einem einzigen Individuum. Zur Erzeugung der neuen Population werden aus den Individuen der aktuellen Generation (diese Individuen werden *Parents* oder *Eltern* genannt) durch einen Rekombinationsoperator sowie einen Mutationsoperator zunächst neue Individuen (die *Offspring* bzw. *Kinder*) erzeugt. Aus der Offspring-Menge werden die Individuen der neuen Generation gewählt (*Selektion*); bei einigen Varianten von Evolutionären Algorithmen wird dabei zusätzlich auch die Menge der Parents berücksichtigt. In Abbildung 7.1 ist dieses Iterationsschema, welches prinzipiell allen Evolutionären Algorithmen zugrunde liegt, dargestellt.

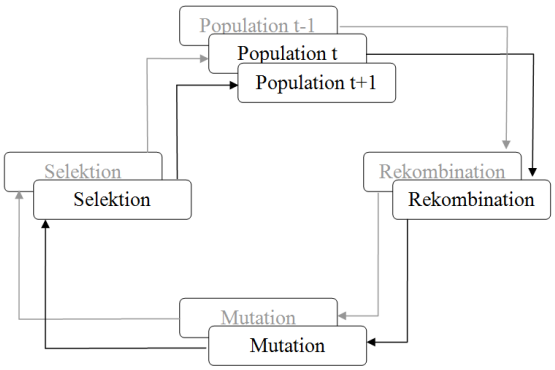


Abbildung 7.1: *Generelles Iterationsschema bei Evolutionären Algorithmen.*

Es gibt unterschiedliche Arten von Evolutionären Algorithmen. *Evolutionsstrategien*[1] gehen auf Rechenberg zurück [Rec73]. Diese sind später insbesondere von Schwefel genauer untersucht und weiterentwickelt worden [Sch75]. Ungefähr zeitgleich zu Rechenberg hat Holland *Genetische Algorithmen* entwickelt [Hol75]. Dabei hat er das Hauptaugenmerk darauf gerichtet, die natürlichen Prozesse möglichst gut abzubilden. Weiterhin zählt auch die *Evolutionäre Programmierung* zu den Evolutionären Algorithmen. Eine vergleichende Beschreibung dieser drei Ansätze ist in [Bäc96] zu finden. Auf Ähnlichkeiten und Unterschiede zwischen Evolutionärer Programmierung und Evolutionsstrategien wird auch in [BRS93] eingegangen.

[1] Abkürzend wird üblicherweise auch *ES* anstelle von *Evolutionsstrategie* geschrieben.

Neben den Evolutionären Algorithmen gibt es weitere naturanaloge Optimierungsverfahren, denen ähnliche Ideen zugrunde liegen. *Simulated Annealing* [KGV83] ist ein bekanntes Beispiel, bei dem jedoch nicht die biologischen Prozesse als Analogie herangezogen, sondern chemische Eigenschaften bei der Abkühlung im Bereich der Metallurgie. Ein wesentlicher Unterschied zwischen Evolutionären Algorithmen und Simulated Annealing besteht darin, dass bei Simulated Annealing lediglich mit einer aktuellen Lösung gearbeitet wird, während Evolutionären Algorithmen häufig Populationen nutzen. Darüber hinaus können bei Simulated Annealing auch schlechtere Lösungen selektiert werden, wobei die Wahrscheinlichkeit dafür aber kontinuierlich sinkt.

Da das EvOpSeg-Verfahren eine Evolutionsstrategie verwendet, wird auf diese Variante der Evolutionären Algorithmen in den folgenden Abschnitten genauer eingegangen. Unterabschnitt 7.1.1 behandelt die so genannten klassischen Evolutionsstrategien. Auf eine etwas neuere Form, gemischt-ganzzahlige Evolutionsstrategien, wird in Unterabschnitt 7.1.2 eingegangen, bevor in Unterabschnitt 7.1.3 interaktive Varianten vorgestellt werden. Unterabschnitt 7.1.4 beschreibt schließlich das Konzept der mehrkriteriellen Optimierung.

7.1.1 Klassische Evolutionsstrategie

Wie bereits erwähnt gehen *Evolutionsstrategien* auf Rechenberg zurück [Rec73]. Sie basieren auf einer Population von Individuen, von denen jedes einzelne ein Element des Raums aller möglichen Lösungen ist. Üblicherweise werden bei Evolutionsstrategien die zu optimierenden Parameter als Vektoren reeller Zahlen kodiert.

Bäck und Schwefel geben in [BS93] einen ausführlichen Überblick über Evolutionäre Algorithmen. Die dortige Notation wird – ergänzt um weitere Bezeichnungen aus [Bäc96] – auch im vorliegenden Abschnitt verwendet. Demnach wird für ein n-dimensionales Problem mit $f : \mathbb{R}^n \rightarrow \mathbb{R}$ die *Zielfunktion* bezeichnet, für die eine optimale Parameterbelegung zu ermitteln ist, so dass der Funktionswert je nach Problemstellung maximiert oder minimiert wird. Im ersten Fall spricht man von einem *Maximierungsproblem*, im zweiten entsprechend von einem *Minimierungsproblem*. Auch wenn sich ein Minimierungsproblem durch Multiplikation der Zielfunktion mit -1 in ein Maximierungsproblem (und umgekehrt) überführen lässt, findet man in der Literatur recht häufig Minimierungsprobleme. Aus diesem Grund wird auch im Folgenden stets von einem Minimierungsproblem ausgegangen.

Jedes Individuum besteht zumindest aus einem Vektor von reellen Zahlen, der für eine konkrete Belegung der Parameter steht und als *Vektor von Objektvariablen* bezeichnet wird. Dieser Vektor wird auch *Chromosom* genannt und jede einzelne seiner Komponenten *Gen*. Außer den Werten der Parameter können die Individuen aber auch noch weitere Elemente enthalten, die eine Belegung der bei der Mutation verwendeten Strategieparameter und damit ein weiteres Chromosom bilden. Das ist bei ES-Varianten sinnvoll, die eine selbstadaptierende Schrittweitenanpassung beinhalten. Darauf wird weiter hinten in diesem Abschnitt noch näher eingegangen.

Die Menge aller möglichen Individuen wird mit I bezeichnet. Der Fitnesswert eines Individuums $\vec{a} \in I$ ist bei Evolutionsstrategien durch den Zielfunktionswert des Objektvariablenvektors von $\vec{a}$ gegeben, d.h. die *Fitnessfunktion* $\Phi : I \to \mathbb{R}$ ist definiert durch $\Phi(\vec{a}) := f(\vec{x})$, wobei $\vec{x} \in \mathbb{R}^n$ der Vektor der im Individuum $\vec{a}$ enthaltenen Objektvariablen ist. Bei Genetischen Algorithmen brauchen diese beiden Funktionen hingegen nicht übereinzustimmen. Eine Population der Generation t besteht aus μ Individuen. Formal wird sie als $P(t) = \{\vec{a}_1, \ldots, \vec{a}_\mu\}$ mit $\vec{a}_i \in I$ notiert.

Aus den Individuen dieser Population werden zunächst durch *Rekombination* λ neue Individuen erzeugt[2]. Der *Rekombinationsoperator* wird mit $r_{\Theta_r} : I^\mu \to I^\lambda$ bezeichnet[3]. Nach [Sch95] sollte der nicht zwangsläufig ganzzahlige Quotient $\frac{\lambda}{\mu}$ im Intervall $[5, 7]$ liegen. Die durch die Rekombination entstandenen Individuen werden anschließend mutiert und bilden dann die so genannten Offspring. Formal wird der *Mutationsoperator* als $m_{\Theta_m} : I^\lambda \to I^\lambda$ notiert. Die beiden Operatoren r_{Θ_r} und m_{Θ_m} dienen der Erzeugung aller λ Individuen. In der Regel lassen sich jedoch auch lokale Operatoren $r'_{\Theta_r} : I^\mu \to I$ und $m'_{\Theta_m} : I \to I$ ableiten, die nur ein einziges Individuum erzeugen[4] [BS93]. Gelegentlich werden der Rekombinationsoperator und der Mutationsoperator auch zu einem einzigen *Variationsoperator* zusammengefasst [BHN+99].

Nach der Mutation wird für jedes der Offspring-Individuen dessen Fitnesswert berechnet. Anschließend erfolgt anhand dieser Fitnesswerte die Auswahl der Individuen, die als Parents der nachfolgenden Generation genommen werden (*Selektion*). Abhängig von der Wahl einer speziellen Variante der Evolutionsstrategien werden dabei entweder sowohl alle Parent-Individuen als auch alle Offspring-Individuen berücksichtigt (diese Variante ist als (μ+λ)-ES oder *Plus-Strategie* bekannt), ausschließlich die Offspring-Individuen ((μ,λ)-ES oder *Komma-Strategie*) oder aber alle Individuen, die ein gewisses Alter κ noch nicht erreicht haben. Letztere Variante wurde in [Sch95] als (μ,κ,λ)-ES eingeführt. Bei der (μ+λ)-ES und der (μ,λ)-ES handelt es sich um Spezialfälle der (μ,κ,λ)-ES. Für $\kappa = 1$ ergibt sich die Komma-Strategie, für $\kappa = \infty$ die Plus-Strategie.

Der Pseudocode einer (μ,κ,λ)-Evolutionsstrategie ist in Listing 7.1 zu sehen (vgl. auch [EGSG00]). Die while-Schleife wird solange durchlaufen, bis ein *Terminierungskriterium* erfüllt ist. Mögliche Beispiele für ein solches Kriterium sind die Vorgabe einer maximalen Anzahl von Generationen oder einer Zeitdauer, nach der die Optimierung beendet wird. Aber auch der Optimierungsfortschritt kann genommen werden, d.h. sobald über eine längere Zeit keine Verbesserung eintritt, wird die while-Schleife verlassen. Das Terminierungskriterium wird mit $\iota : I^\mu \to \{true, false\}$ bezeichnet [BS93].

[2] Weiter unten werden eine Komma-Strategie und eine Plus-Strategie definiert. Bei der Komma-Strategie gilt stets $\lambda \geq \mu$ und bei der Plus-Strategie $\lambda \geq 1$.

[3] Der Index Θ_r gibt an, dass eine Menge von Rekombinationsparametern einfließen kann. Entsprechende Parametermengen Θ_s und Θ_m gibt es auch für die Selektion und die Mutation. Für weitere Details sei auf [Bäc96] verwiesen.

[4] Weit verbreitet ist eine so genannte geschlechtliche Rekombination, bei der aus zwei Individuen ein neues erzeugt wird. In dem Fall ist $r'_{\Theta_r} : I^2 \to I$ (vgl. [Bäc96, S. 74]).

```
1  t := 0
2  Initialisierung der Population P(t) ∈ I^μ
3  Auswertung der Zielfunktion für die μ initialen Individuen
4  while Terminierungskriterium ist nicht erfüllt do
5  {
6    Erzeugung von λ vielen Offspring aus den μ Parents.
7    Initialisierung des Alters dieser Offspring mit 0.
8    Mutation der λ Offspring.
9    Auswertung der Zielfunktion für die λ Offspring.
10   Selektion der μ besten Individuen aus der Menge aller
11     Individuen, die das Maximalalter κ noch nicht
12     erreicht haben. ⇒ Population P(t+1)
13   Erhöhung des Alters jedes Individuums aus P(t+1) um 1.
14   t := t+1
15 }
```

Listing 7.1: *Pseudocode einer (μ, κ, λ)-Evolutionsstrategie.*

Wie bereits erwähnt können Individuen neben den Objektvariablen auch Strategieparameter enthalten. Nach [BS93] und [Bäc96] lässt sich ein Individuum dann als $\vec{a} = (\vec{x}, \vec{\sigma}, \vec{\alpha})$ darstellen, wobei die reellwertige Elemente beinhaltenden Vektoren $\vec{\sigma}$ und $\vec{\alpha}$ die *Strategieparameter* sind. Die Elemente von $\vec{\sigma}$ geben an, wie groß die Streuung der Zufallszahlen bei der Mutation ist; hierauf wird weiter hinten in diesem Abschnitt noch näher eingegangen. Die Elemente von $\vec{\alpha}$ werden *Rotationswinkel* genannt. Sie sind notwendig, wenn eine korrelierte Mutation (vgl. [Sch81, S. 239-243]) erfolgen soll, d.h. wenn die Mutationen der einzelnen Parameterwerte nicht unabhängig voneinander erfolgen. Bei der in dieser Arbeit betrachteten Patch-Optimierung ist jedoch keine derartige Korrelation der Mutationen gegeben, so dass keine Rotationswinkel verwendet werden.

Rekombination

Bei der *Rekombination* wird aus der Menge der Parent-Individuen, die nichts anderes als die aktuelle Population ist, eine Menge von λ vielen neuen Individuen erzeugt. Es gibt verschiedene Rekombinationsformen. Als erstes ist die *ungeschlechtliche Rekombination* zu nennen, bei der lediglich Kopien der zufällig ausgewählten Parent-Individuen angefertigt werden. Im eigentlichen Sinne findet dabei keine wirkliche Rekombination statt. Da bei Evolutionsstrategien der Mutationsoperator der entscheidende Operator ist und der Rekombination eine eher geringe Bedeutung zukommt [SHF94, Ort02] wird gelegentlich eine ungeschlechtliche Rekombination verwendet. Bei allen weiteren Rekombinationsvarianten werden (mindestens) zwei Parent-Individuen zur Erzeugung eines neuen Individuums herangezogen. Man spricht dann auch von einer *geschlechtlichen Rekombination* bzw. bei mehr als zwei Eltern von einer *panmiktischen Rekombination*. Die Auswahl der Parent-Individuen, die zur Rekombination herangezogen werden, ist auch als *Paarungsselektion* bekannt.

Bei einer geschlechtlichen Rekombination werden zunächst zwei Parent-Individuen S und T zufällig gewählt. Es sei S_i das i-te Gen von S und T_i das i-te Gen von T. Bei der lokalen diskreten Rekombination wird für alle $i \in \{1, \ldots, w\}$, wobei w die Anzahl der

Elemente eines Individuums ist, zufällig entweder S_i oder T_i gewählt. Bei der lokalen intermediären Rekombination ergibt sich der i-te Wert des neuen Individuums hingegen nach $(1-\chi)\cdot S_i+\chi\cdot T_i$. Dabei ist entweder $\chi \in [0,1]$ eine gleichverteilte Zufallsvariable, die für jeden Parameter neu berechnet wird, oder es gilt stets $\chi = 0.5$. Letzteres findet in der vorliegenden Arbeit Anwendung.

Die panmiktischen Varianten der diskreten bzw. der intermediären Rekombination unterscheiden sich von den soeben beschriebenen geschlechtlichen Varianten lediglich dadurch, dass im panmiktischen Fall für jedes der w Elemente des Individuums S ein neues Partner-Individuum T aus der Menge der Parent-Individuen zufällig gewählt wird. Außerdem wird auch χ für jedes Element von S neu ermittelt. Eine ausführliche Beschreibung der unterschiedlichen Rekombinationsvarianten findet sich beispielsweise in [Bäc96, S. 74].

Es ist nicht zwingend erforderlich, eine einzige Rekombinationstechnik für alle Parameter zu verwenden: Stattdessen kann beispielsweise für die Strategieparameter eine andere Technik gewählt werden als für die Objektvariablen.

Mutation

Die Mutation eines Individuums bewirkt, dass tendenziell kleine Änderungen an dessen Chromosom durchgeführt werden. Im Gegensatz zu vorherigen Ansätzen, bei denen wie in [Sch95] beschrieben im Durchschnitt nur eine einzige Komponente verändert wird, erfolgt bei Evolutionsstrategien in jedem Mutationsschritt eine Veränderung aller Komponenten des Vektors. Bei der Mutation eines Individuums $\vec{a}$ wird zu jeder Komponente von $\vec{a}$ eine normalverteilte Zufallszahl mit Erwartungswert 0 und einer gegebenen Standardabweichung σ addiert. Durch σ wird die Mutationsschrittweite gesteuert. Ist der Wert zu klein gewählt, kann die Anzahl der benötigten Generationen und damit auch die Rechenzeit beliebig groß werden [GKK04, S. 116]. Bei einem zu großen σ besteht die Gefahr, dass die Individuen der neuen Generation kaum noch Ähnlichkeiten zu ihren Eltern haben und somit eher zufällig erzeugt werden. Nur ein kleiner Bereich um das optimale σ (also die optimale Schrittweite) herum garantiert eine gute Konvergenzgeschwindigkeit. Rechenberg hat diesen Bereich als *Evolutionsfenster* bezeichnet [Rec73].

Es haben sich unterschiedliche Ansätze entwickelt, wie mit diesem Problem umgegangen werden kann. Eine Möglichkeit ist die *1/5-Erfolgsregel*, die Rechenberg anhand von zwei Funktionen (dem Korridormodell und dem Kugelmodell) für eine (1+1)-Evolutionsstrategie entwickelt hat [Rec73]. Danach ist die Mutationsschrittweite optimal gewählt, wenn etwa 20 Prozent der Mutationen erfolgreich sind. Dazu wird die Rate nach n Mutationen überprüft, wobei jedoch mehrere Mutationen, zum Beispiel die letzten $10n$ Stück, berücksichtigt werden [Bäc96]. Falls weniger als 20 Prozent dieser Mutationen erfolgreich waren, wird σ mit einem $c \in \mathbb{R}^+$ multipliziert. Falls mehr als 20 Prozent der Mutationen erfolgreich waren, wird σ durch c dividiert. Nach [Bäc96] sollte c im Intervall $[0.817, 1]$ liegen, Schwefel schlägt in [Sch95] den Wert $c = 0.85$ vor. Die 1/5-Regel kann unter gewissen Umständen zu einer permanenten Reduktion

der Mutationsschrittweite und somit zu einer vorzeitigen Konvergenz führen [GKK04, S. 118].

Die Anpassung der Mutationsschrittweiten bei der 1/5-Regel erfolgt gewissermaßen von „außerhalb“ des evolutionären Prozesses und entspricht somit nicht dem biologischen Paradigma, welches einer Evolutionsstrategie zugrunde liegt [Sch95, S. 142]. Des Weiteren erfolgt die Anpassung unabhängig von der konkreten Problemstellung (vgl. auch [SHF94, S. 179]). Abhilfe schafft die so genannte *Selbstadaption*, bei der die Mutationsschrittweite, d.h. die Standardabweichung σ, als zu optimierender *Strategieparameter* zusätzlich mit in das Individuum aufgenommen wird. Die Idee dabei ist, dass auch die Mutionsschrittweite σ mutiert wird und sich gute Schrittweiten durch Selektion automatisch im Laufe der Zeit durchsetzen, ohne dass sie jedoch einen direkten Einfluss auf die Fitnessfunktion haben.

Bei der Selbstadaption ist zu unterscheiden, ob eine einzige Mutationsschrittweite σ für alle Parameter verwendet wird [Sch95, S. 143], oder ob eine *Mutation mit n Einzelschrittweiten* vorliegt [Sch95, S. 144]. Hebbel spricht im ersten Fall, bei dem das Individuum die Gestalt

$$\vec{a} = (x_1, \ldots, x_n, \sigma) \tag{7.1}$$

hat, von einer *einfachen Mutation* [Heb09, S. 38]. Der Strategieparameter σ wird dabei lognormalverteilt mutiert, um so negative Schrittweiten auszuschließen (vgl. [Heb09, S. 38]). Es ergibt sich durch die Mutation also $\sigma_{neu} = c \cdot \sigma$ mit einem $N(0, \tau^2)$-verteilten $\ln c$. Dabei sollte bei n Objektvariablen gelten: $\tau \propto \frac{1}{\sqrt{n}}$ (vgl. beispielsweise auch [Bäc96, S. 72]).

Im Falle einer Selbstadaption mit n Einzelschrittweiten wird zu jeder Objektvariablen x_i ein zusätzlicher Strategieparameter $\sigma_i \in \mathbb{R}$ in das Individuum aufgenommen, so dass dieses schließlich die Gestalt

$$\vec{a} = (x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n) \tag{7.2}$$

hat (vgl. auch [GKK04]). An der Mutation der Parameter x_1 bis x_n ändert sich vom Prinzip her nichts, außer dass nun für jeden Parameter eine eigene Standardabweichung genommen wird. Auch die Strategieparameter σ_1 bis σ_n werden einzeln mutiert. Für $i = 1, \ldots, n$ ergibt die Mutation

$$\sigma_i^{mutiert} = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)) \tag{7.3}$$

mit

$$\tau \approx \frac{1}{\sqrt{2\sqrt{n}}}$$

$$\tau' \approx \frac{1}{\sqrt{2n}}$$

wobei $N(0,1)$ und $N_i(0,1)$ normalverteilte Zufallsvariablen mit Erwartungswert 0 und Standardabweichung 1 sind, die sich darin unterscheiden, dass $N_i(0,1)$ für jedes i neu berechnet wird [Sch81, BS93, GKK04].

Selektion

Bei Evolutionsstrategien werden stets die Individuen mit den besten Fitnesswerten als neue Parent-Individuen gewählt, d.h. die Selektion verläuft komplett deterministisch [BS93], zumindest sofern es keine Individuen mit identischen Fitnesswerten gibt. Selektiert wird bei einer (μ, κ, λ)-Evolutionsstrategie aus der Menge aller Individuen, die das Maximalalter κ noch nicht erreicht haben[5]. Wie weiter oben bereits erwähnt, beinhaltet eine (μ, κ, λ)-Evolutionsstrategie sowohl Eigenschaften einer Plus-Strategie als auch einer Komma-Strategie, und kann durch Setzen von $\kappa = 1$ bzw. $\kappa = \infty$ zu einer Komma-Strategie bzw. einer Plus-Strategie degenerieren. Nach [BS93] gefährdet eine Plus-Strategie im Gegensatz zu einer Komma-Strategie den Erfolg einer Selbstadaption der Strategieparameter. Bei einer Komma-Strategie kann es jedoch dazu kommen, dass alle Offspring-Individuen schlechtere Fitnesswerte haben als das beste Parent-Individuum und somit gute Lösungen verloren gehen. Um das zu verhindern kann bei einer Komma-Strategie stets die beste aller bisher gefundenen Lösungen zusätzlich gespeichert werden, wenngleich sie bei der Selektion nicht berücksichtigt wird [GKK04, S. 116]. Dies ist auch bei einer (μ, κ, λ)-Evolutionsstrategie zu empfehlen, bei der das beste Individuum jedoch noch für eine gewisse Zeit für die Selektion in Frage kommt.

In Abbildung 7.2 sind die Auswirkungen der drei angesprochenen Selektionsarten für ein Minimierungsproblem veranschaulicht. Das obere Teilbild bezieht sich auf die Komma-Strategie sowie auf die Plus-Strategie. Die Figuren symbolisieren Individuen, die Zahlen stehen für die Fitnesswerte der Individuen. Im unteren Teilbild ist die (μ, κ, λ)-Strategie veranschaulicht. Es liegen die gleichen Parent- und Offspring-Individuen vor wie im oberen Teilbild. Für die Individuen wird jedoch zusätzlich das Alter (in Anzahl Generationen) vermerkt, was mit Hilfe von Schildern an den Figuren dargestellt ist. Es können zwar Individuen aus der Elterngeneration selektiert werden, jedoch hängt dies von deren Alterswerten ab. So hat das Individuum mit dem kleinsten Fitnesswert in der i-ten Generation das Maximalalter bereits erreicht und wird nicht als Eltern-Individuum in die nachfolgende Generation übernommen. Dabei ist $\kappa = 3$ zugrunde gelegt worden.

7.1.2 Gemischt-ganzzahlige Evolutionsstrategien

Klassische Evolutionsstrategien nutzen Individuen, deren Parameter ausschließlich reelle Zahlen sind. *Gemischt-ganzzahlige Evolutionsstrategien*, die auch *Mixed-Integer-Evolutionsstrategien* genannt werden (kurz *MI-ES*), sind speziell angepasste Evolutionsstrategien, die neben reellen Parametern auch ganzzahlige sowie nominal-diskrete Parameter verarbeiten können [Sch96, EGSG00]. Wird in einem Parameter beispielsweise die Anzahl bestimmter Elemente oder Dinge gespeichert, so handelt es sich um einen ganzzahligen Parameter. Bei nominal-diskreten Parametern kann keine wirkliche Ordnung definiert werden (vgl. [EGSG00]). Somit reicht bei der Mutation solcher

[5] Wobei die Offspring-Individuen zu diesem Zeitpunkt das Alter 0 haben.

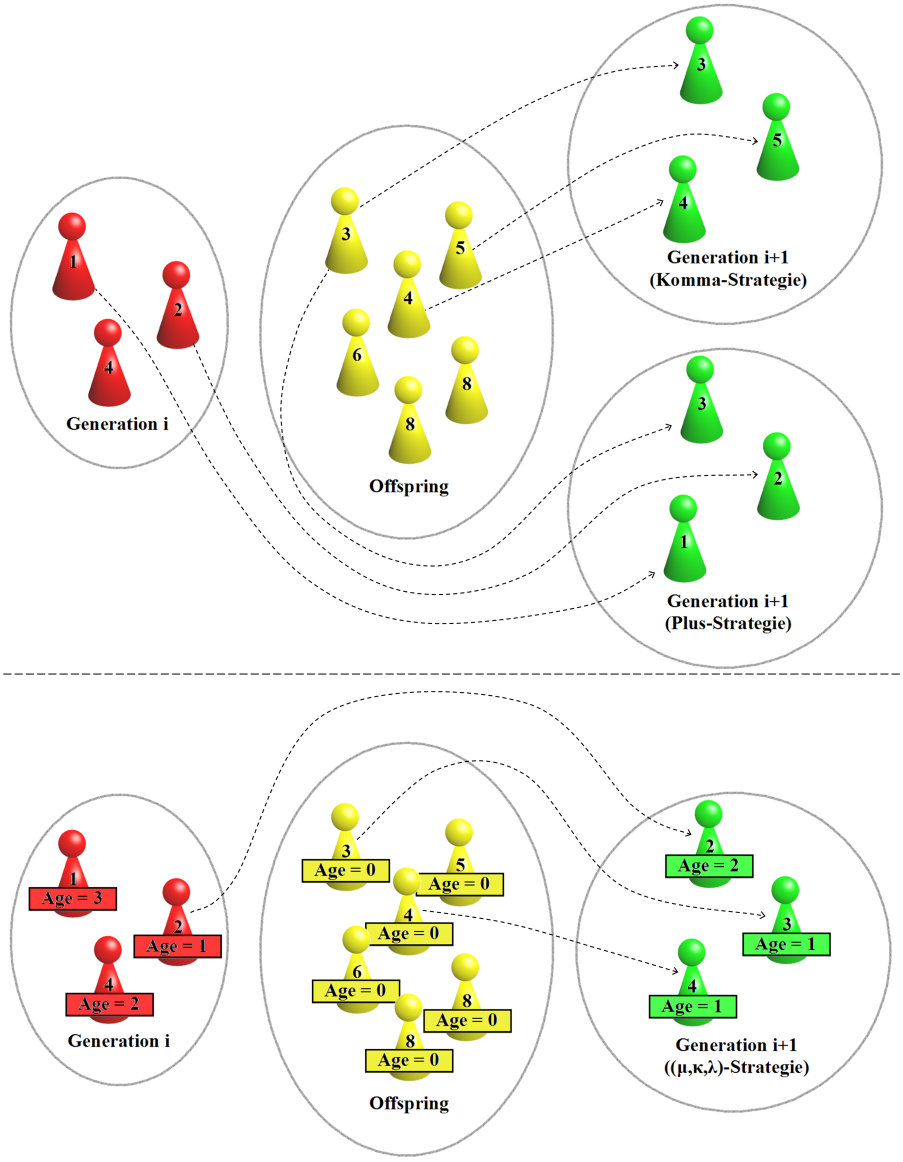


Abbildung 7.2: *Vergleich der drei gängigen Selektionsarten. Bei der* (μ, κ, λ)*-Strategie ist dabei* $\kappa = 3$ *gewählt worden.*

Parameter auch keine simple Addition einer normalverteilten Zufallszahl aus. Stattdessen ist eine neue, an die konkrete Problemstellung angepasste Mutationsvariante zu realisieren. Ein nominal-diskreter Parameter kann beispielsweise angeben, welche von mehreren gleichwertigen Alternativen gewählt wird. In einem solchen Fall wird bei der Mutation eine Alternative zufällig gewählt, wobei eine Gleichverteilung zugrunde liegt [EGSG00]. Durch die Angabe einer Mutationswahrscheinlichkeit für den Parameter [Sch96] kann dabei erreicht werden, dass solche Mutationen nicht zu häufig auftreten.

7.1.3 Interaktive Evolutionsstrategien

Das EvOpSeg-Verfahren ist zwar weitgehend als automatisches Segmentierungsverfahren gedacht, es soll jedoch auch sinnvolle interaktive Eingriffe durch die Anwendenden ermöglichen. *Interaktive Evolutionsstrategien* [BEB06] bieten unterschiedliche Arten von Eingriffsmöglichkeiten seitens der Anwendenden. Es wird zwischen *reaktiven Interaktionen* und *proaktiven Interaktionen* unterschieden [BEB06]. Bei Optimierungsverfahren, die auf reaktive Interaktionen ausgelegt sind, nimmt ein menschlicher Akteur bspw. die Bewertung oder die Selektion der Individuen vor. Bei proaktiven Interaktionen können hingegen optional einzelne Parameter oder auch Individuen angepasst werden, ohne dass dies vom Algorithmus explizit gefordert wird.

Reaktive Interaktion

Bei reaktiver Interaktion sind Benutzereingriffe zwingend erforderlich. Im Kontext der Evolutionären Algorithmen versteht man darunter in erster Linie, dass die Fitnessauswertung nicht automatisch erfolgt, sondern durch Menschen vorgenommen wird. Takagi gibt in [Tak98] einen umfassenden Überblick über reaktive Interaktionen und nennt diese Form der Optimierung *Interactive Evolutionary Computation*. Nach Takagi bewerten die Menschen ausschließlich das Ergebnis, das sich für die Parameterwerte des jeweiligen Individuums ergibt, und nicht etwa die Parameterwerte als solche. Diese Variante bietet sich insbesondere bei visuellen sowie auditiven Problemstellungen an, bei denen ein Bild oder eine Melodie zu optimieren sind. Takagi weist in [Tak98] auf folgende Eigenschaften der Interactive Evolutionary Computation hin:

1. Die subjektive Wahrnehmung der Menschen kann sich im Laufe der Zeit ändern. Wenn dasselbe zu bewertende Bild oder dieselbe Melodie einem Menschen wiederholt präsentiert wird, kann die zweite Bewertung durchaus anders ausfallen als zuvor.

2. Interactive Evolutionary Computation tendiert dazu, nicht gegen ein einziges Optimum sondern gegen eine Region zu konvergieren, deren Elemente von Menschen als gleichwertig angesehen werden. Breukelaer et al. [BEB06] geben dazu als Beispiel ein RGB-Color-Redesign-Problem an, bei dem die RGB-Werte einer

gegebenen Farbe möglichst genau zu ermitteln sind. Die Anwendenden selektieren das Individuum, dessen zugehörige Farbe sie als der Zielfarbe am ähnlichsten ansehen. Allerdings können Menschen nicht beliebig feine Farbunterschiede optisch wahrnehmen[6], wodurch sie im Gegensatz zu einem Computer nicht beliebig kleine Qualitätsunterschiede erkennen und somit auch nicht vollkommen optimal selektieren können.

3. Auch wenn die Bestrebungen dahin gehen, den Interaktionsaufwand zu minimieren, müssen die Anwendenden dennoch zahlreiche Bewertungen von Individuen vornehmen. Dies ist eine ermüdende Tätigkeit, die dazu führt, dass sie mit fortschreitender Dauer nicht mehr so aufmerksam vorgehen wie zu Beginn. Die Zeit, die für die manuelle Bewertung benötigt wird, kann zwar in einigen Domänen bereits nach dem Pipelining-Prinzip für andere Berechnungsschritte genutzt werden, dennoch geht die manuelle Bewertung aber immer mit einem gewissen Geschwindigkeitsverlust einher.

Proaktive Interaktionen

Im Gegensatz zu reaktiven Interaktionen sind proaktive Interaktionen optional, so dass das Optimierungsverfahren im Wesentlichen automatisch abläuft. Dadurch müssen die Anwendenden insgesamt deutlich seltener Einfluss nehmen als bei einem auf reaktiven Interaktionen basierenden Verfahren. Der einfachste Fall einer proaktiven Interaktion ist durch das manuelle Beenden eines Optimierungslaufs gegeben, beispielsweise durch Betätigung eines entsprechenden Button. Auch die Anpassung zentraler Parameter wie der Populationsgröße oder der Schrittweite ist möglich. Dazu ist die Optimierung zunächst anzuhalten. Mit den geänderten Werten kann die Optimierung anschließend fortgesetzt werden.

Interaktionen bei der Optimierung im Rahmen des EvOpSeg-Verfahrens

Wie schon in den einleitenden Kapiteln erwähnt, lässt sich im Allgemeinen nicht eindeutig angeben, wann eine Segmentierung als optimal anzusehen ist. Dieser Aspekt könnte dafür sprechen, die Fitnessbewertung reaktiv vornehmen zu lassen, da die formale Angabe einer Optimalitätsbedingung offensichtlich nicht allgemeingültig möglich ist. Die Durchführung einer solchen reaktiven Bewertung ist jedoch einerseits ermüdend und andererseits auch recht zeitaufwändig [Tak98]. Für jedes Individuum müssten sich die Anwendenden das Modell von allen Seiten ansehen, es also komplett im Raum drehen oder entsprechende Ansichten als Bilder präsentiert bekommen. Insbesondere beim Drehen des Modells ist es oft nicht einfach, einen guten Überblick über alle Segmentgrenzen zu behalten und die Segmentierung verglichen mit den Segmentierungen

[6] Menschen können beispielsweise den Unterschied der Farbtöne RGB(200,200,0) und RGB(200,199,0) nicht wirklich wahrnehmen.

anderer Individuen bzw. früherer Generationen angemessen zu bewerten. Aus den genannten Gründen ist bei der Evolutionsstrategie des EvOpSeg-Verfahren von reaktiven Interaktionen Abstand genommen worden.

Beim EvOpSeg-Verfahren kommen proaktive Interaktionsmöglichkeiten zum Einsatz. Die Anwendenden können die evolutionäre Optimierung anhalten und sämtliche konfigurierbaren Parameter adaptieren. Dazu wird den Anwendenden die zu dem bis dahin am besten bewerteten Individuum gehörende Segmentierung präsentiert. Da nach dem Stoppen der Optimierung durchaus auch patchverändernde Interaktionen (vgl. Abschnitt 5.3) durchgeführt werden können, die einen anderen Aufbau der Individuen bedingen können, wird beim erneuten Starten des Optimierungslaufs wieder eine neue, homogene Startpopulation erzeugt.

7.1.4 Mehrkriterielle Optimierung

Die in Abschnitt 7.1.1 beschriebene klassische Evolutionsstrategie dient der Ermittlung einer möglichst optimalen Parameterbelegung für eine Funktion, deren Wertebereich der Raum $\mathbb{R}$ der reellen Zahlen oder ein Unterraum von diesem ist. Betrachtet wird also lediglich ein einziges „Qualitätskriterium“. Häufig beinhalten Problemstellungen jedoch mehrere Qualitätskriterien, die unabhängig voneinander sind. Jedem einzelnen kommt dabei eine nicht unwesentliche Bedeutung zu, so dass eine zumindest nahezu optimale Gesamtlösung möglichst gut in Bezug auf jedes einzelnes Kriterium sein sollte. Mitunter stehen aber einige der Qualitätskriterien in einem Konflikt miteinander. Ein anschauliches Beispiel stellt der Kauf einer HiFi-Anlage dar. Einerseits möchte man eine möglichst gute Klangqualität erhalten, andererseits aber auch möglichst wenig Geld investieren. Damit liegen zwei Ziele vor, gegen die zu optimieren ist, die aber einander widersprechen: Die günstigste HiFi-Anlage wird nicht den besten Klang liefern und eine Anlage mit guter Klangqualität wird nicht gerade kostengünstig sein. Dies ist auch in Abbildung 7.3 anhand eines hypothetischen Beispiels zu erkennen, bei dem die Klangqualität gegen den Preis aufgetragen ist. Zu beachten ist, dass es sich bei der Klangqualität um ein Maximierungsproblem handelt, während die Suche nach einem günstigen Preis ein Minimierungsproblem darstellt.

Im Kontext eines mehrkriteriellen Optimierungsproblems spricht man davon, dass eine Lösung $\vec{x}$ eine andere $\vec{x}'$ *dominiert*, wenn die beiden folgenden Bedingungen erfüllt sind [Deb02, S. 28]:

1. Es gibt kein Kriterium, bezüglich dessen die Lösung $\vec{x}$ schlechter ist als die Lösung $\vec{x}'$.
2. Die Lösung $\vec{x}$ ist zumindest in Bezug auf ein Kriterium echt besser als die Lösung $\vec{x}'$.

Jedes Kriterium wird durch eine eigene Zielfunktion dargestellt. Nach [Deb02, S. 31] ist für eine gegebene Lösungsmenge P die Menge der *nicht-dominierten Lösungen* P'

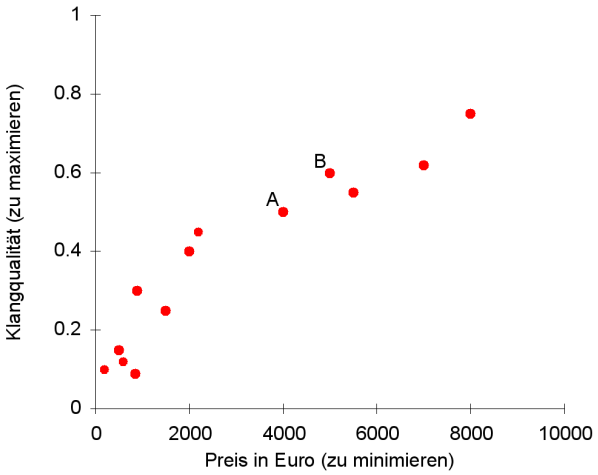


Abbildung 7.3: *Hypothetische Lösungen für ein mehrkriterielles Optimierungsproblem am Beispiel des Kaufs einer HiFi-Anlage.*

definiert als die Menge aller Elemente von P, die von keinem Element aus P dominiert werden. Die Gesamtheit aller überhaupt möglichen, nicht-dominierten Lösungen wird auch Menge der *pareto-optimalen Lösungen* genannt [Deb02, S. 31]. In Abbildung 7.4 ist die Menge der Zielfunktionsergebnisse aller möglichen Lösungen eines zweikriteriellen Optimierungsproblems mit den beiden „Teil-Zielfunktionen“ f_1 und f_2 als graue Fläche dargestellt. Analog zu dem HiFi-Beispiel sei auch hier f_1 zu minimieren und f_2 zu maximieren. Die Funktionsergebnisse der pareto-optimalen Lösungen sind in der Abbildung durch eine rote Linie markiert. Diese Linie stellt die so genannte *Paretofront* dar.

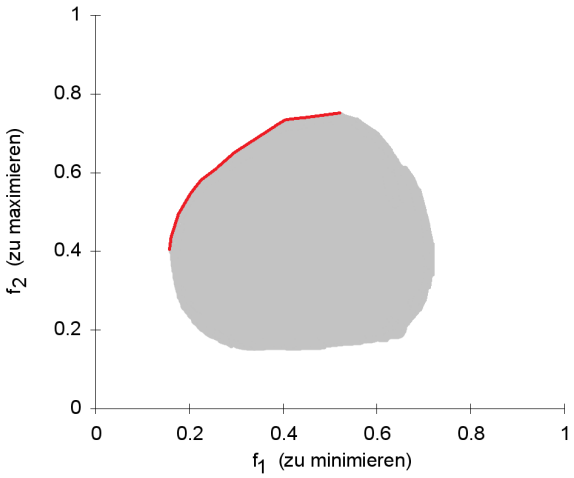


Abbildung 7.4: *Veranschaulichung der Paretofront für ein mehrkriterielles Optimierungsproblem, bei dem f_1 zu minimieren und f_2 zu maximieren ist.*

Bei einem mehrkriteriellen Optimierungsproblem sind manche Lösungen, insbesondere die pareto-optimalen, nicht miteinander vergleichbar; beispielsweise ist der Preis der

zu der in Abbildung 7.3 mit A markierten Lösung gehörenden HiFi-Anlage besser als der einer zweiten Anlage B, wohingegen diese aber eine deutlich bessere Klangqualität liefert als die erste. Die Beurteilung, welche der beiden genannten Lösungen zu präferieren ist, kann nur mit Hilfe weiterer externer Informationen (wie beispielsweise den zur Verfügung stehenden finanziellen Mitteln) erfolgen.

Deb nennt in [Deb02] zwei grundlegend verschiedene Vorgehensweisen zur Behandlung von mehrkriteriellen Optimierungsproblemen. Die erste bezeichnet er als *ideal*. Bei dieser werden zunächst mehrere pareto-optimale Lösungen gesucht, d.h. es wird gegen die Paretofront optimiert. Erst im Anschluss wird eine dieser pareto-optimalen Lösungen als Gesamtlösung gewählt. Dazu sind allerdings, wie schon oben erwähnt, weitere Informationen erforderlich. Bei der zweiten Vorgehensweise wird hingegen ein mehrkriterielles Optimierungsproblem in ein einkriterielles Problem überführt, das nur noch eine einzige Zielfunktion beinhaltet. Diese Überführung kann beispielsweise mit der *Weighted-Sum-Methode* erfolgen [Deb02, S. 50ff], bei der sich die neue Zielfunktion als gewichtete Summe der ursprünglichen Zielfunktionen ergibt. An dieser Stelle sind für die Konstruktion der neuen Zielfunktion weitere problemspezifische Informationen wie beispielsweise die Gewichtungen der ursprünglichen Kriterien notwendig. Die Gewichtung gibt den Einfluss der einzelnen Zielfunktionen an. Häufig erfolgt jedoch zuvor noch eine Normierung, um die einzelnen Zielfunktionen auf einen gemeinsamen Wertebereich abzubilden. Bei stark unterschiedlichen Wertebereichen wie denen aus Abbildung 7.3 ist eine solche Normierung im Prinzip unabdingbar.

7.2 Optimierung der Patches mit einer Evolutionsstrategie

Obwohl sie einige ähnliche Grundprinzipien aufweisen, wurde von Anfang an klar zwischen Evolutionsstrategien und Genetischen Algorithmen unterschieden. So stellt beispielsweise Bäck in [Bäc96] heraus, dass die Individuen bei Genetischen Algorithmen stets als Binärvektoren repräsentiert werden, wohingegen bei Evolutionsstrategien Vektoren reeller Zahlen vorliegen. Diese beiden Arten von Evolutionären Algorithmen unterscheiden sich auch bezüglich der Bedeutung von Rekombination und Mutation voneinander: Bei Genetischen Algorithmen spielt die Rekombination eine zentrale Rolle, wohingegen der Einfluss der Mutation nahezu vernachlässigt wird [Bäc96]. Im Gegensatz dazu kommt der Mutation bei Evolutionsstrategien eine wesentlich größere Bedeutung als der Rekombination zu [BS93, Bäc96].

Seit Mitte der 1990er Jahre verschwimmen die Grenzen zwischen Evolutionsstrategien und Genetischen Algorithmen zunehmend (vgl. [Nis94, S. 193]). Teilweise findet sogar überhaupt keine Unterscheidung mehr statt, sondern es wird lediglich von einem „Evolutionären Algorithmus“ gesprochen [BS93][Nis94, S. 191]. Trotzdem orientiert sich die vorliegende Arbeit weitgehend an der klassischen Einteilung der Evolutionären Algorithmen, wie sie insbesondere in [Bäc96] zu finden ist. Demnach sind Evolutionsstrategien (bzw. MI-ES) den Genetischen Algorithmen für eine evolutionäre Optimierung

der Patches aus verschiedenen Gründen vorzuziehen. So eignen sich Evolutionsstrategien bei kontinuierlichen Parametern, wie beispielsweise den Seed-Gewichten oder den Faktoren ζ und η, besser als Genetische Algorithmen (vgl. [LEEB06]). Dies liegt vor allem an der Problemkodierung sowie den Variationsoperatoren. Bei Genetischen Algorithmen kommt dem Rekombinationsoperator eine entscheidende Bedeutung zu. Er kann sein Potential allerdings erst dann richtig entfalten, wenn genügend genetische Variationen in der Population vorhanden sind. Dies ist jedoch bei den zu optimierenden Patches nicht der Fall, da lediglich eine einzige Ausgangssituation – das Ergebnis der Initialsegmentierung – und damit eine absolut homogene Startpopulation vorliegt. In solchen Fällen ist ein Optimierungsverfahren zu wählen, bei dem primär die Mutation für den Fortschritt sorgt. Aus den genannten Gründen wird zur Patch-Optimierung eine Evolutionsstrategie, oder genauer formuliert eine gemischt-ganzzahlige Variante einer (μ, κ, λ)-Evolutionsstrategie, verwendet.

7.2.1 Problemkodierung

Unter einer *Problemkodierung* versteht man die Repräsentation von Lösungen der gegebenen Problemstellung. Die zu optimierenden Parameter werden dabei geeignet als *Vektor von Objektvariablen* angegeben. Wie in Abschnitt 7.1 beschrieben ist dieser Vektor ein Bestandteil eines jeden Individuums. Er enthält unter anderem einen binären Parameter $\nu \in \{0, 1\}$, der angibt, ob die in Gleichung (5.2) angegebene gewichtete winkelbasierte ($\nu = 0$) oder die merkmalbasierte Distanz aus Gleichung (5.28) für die Patch-Berechnung zu nehmen ist ($\nu = 1$). Die Wahl der Funktion wird mit in den Optimierungsprozess aufgenommen, da Experimente gezeigt haben, dass bei manchen Modellen bzw. Seed-Konstellationen die merkmalbasierte und bei anderen die winkelbasierte Distanzfunktion bessere Ergebnisse liefert. So kann durch die Optimierung die für das jeweilige Modell besser passende der beiden Funktionen selektiert werden. Formal kommt damit dann folgende *kombinierte Distanzfunktion* zum Einsatz:

$$d_{comb}^{\zeta,\eta,\nu}(t_i, t_j) := \underbrace{d_{geo}(t_i, t_j) + \zeta \cdot d_{\measuredangle}(t_i, t_j)}_{d_{ang}^{\zeta}(t_i,t_j)} + \nu \cdot \eta \cdot d_{shape}(t_i, t_j);\ \zeta, \eta \in \mathbb{R}_0^+. \tag{7.4}$$

Entsprechend wird dann in Analogie zu den Gleichungen (5.30) und (5.31) die folgende gewichtete Distanz verwendet:

$$d_{comb,\gamma}(t', t_{seed}, \gamma_{seed}) := \frac{1}{\gamma_{seed}} \cdot d_{comb}^{\zeta,\eta,\nu}(t', t_{seed}). \tag{7.5}$$

Neben dem Parameter ν zur Auswahl der Distanzfunktion stellen bei der Optimierung der Patches auch die Faktoren ζ und η sowie die Positionen und die Gewichte der Seeds zu optimierende Parameter dar. Der Vektor von Objektvariablen ist demnach durch $\vec{x} := (\vec{s}, \zeta, \eta, \nu)$ gegeben. Dabei enthält der Vektor $\vec{s}$ alle Seed-Punkte, die in der Form $\vec{s}_{i,j} := (p_{i,j}, \gamma_{i,j})$ angegeben sind, und zwar derart, dass Seed-Punkte, die zusammen

eine Patch-Gruppe definieren, in $\vec{s}$ aufeinanderfolgend angeordnet sind. $p_{i,j} \in \mathbb{N}_0$ bezeichnet den Index[7] der Dreiecksfläche, die als Seed-Dreieck des j-ten Sub-Patch von P_i genommen wird, wobei P_i die i-te Patch-Gruppe ist. Analog bezeichnet $\gamma_{i,j} \in \mathbb{R}^+$ das entsprechende Seed-Gewicht. Jedes Patch, das keiner Patch-Gruppe angehört, wird dabei als einelementige Patch-Gruppe angesehen. Die Anordnung der Seeds entsprechend der Patch-Gruppen lässt sich durch $\vec{s} = (\vec{s}_1, \vec{s}_2, \ldots, \vec{s}_m)$ ausdrücken, wobei m die Anzahl der Patch-Gruppen und $\vec{s}_i := (\vec{s}_{i,1}, \vec{s}_{i,2}, \ldots, \vec{s}_{i,n_i})$ den Vektor der zur i-ten Patch-Gruppe gehörenden Seed-Punkte bezeichnen. Im Gegensatz zu den Seed-Positionen und -Gewichten haben $\nu \in \{0, 1\}$ sowie $\zeta, \eta \in \mathbb{R}_0^+$ einen globalen Charakter, d.h. sie sind für alle Sub-Patches gleich.

Alle Gewichte $\gamma_{i,j}$ sind positive, ζ und η nicht-negative reelle Zahlen. Bei den Indizes der Dreiecksflächen handelt es sich hingegen um Elemente aus $\mathbb{N}_0$. Dieses Indizes lassen sich bei einem Dreiecksnetz nicht derart anordnen, dass stets aus einem gegebenen Index die Indizes der benachbarten Dreiecksflächen hergeleitet werden können. Demnach handelt es sich bei den Positionen $p_{i,j}$ um nominal-diskrete Variablen, denen keine logische Ordnung zugrunde liegt (vgl. [LEEB06]).

Bei der evolutionären Patch-Optimierung wird eine einzige Mutationsschrittweite für alle Parameter verwendet. Jedes Individuum enthält somit entsprechend Formel (7.1) nur einen einzigen Strategieparameter σ. Durch die Aufnahme von σ in das Individuum wird eine selbstregulierende Schrittweitenapassung ermöglicht. Auf diesen Strategieparameter wird auch in Unterabschnitt 7.2.3 näher eingegangen.

7.2.2 Erzeugung einer Startpopulation

Im Rahmen der evolutionären Optimierung wird zwischen homogenen und heterogenen Startpopulationen unterschieden. Wenn alle Individuen einer Population möglichst gleichmäßig über den Lösungsraum verteilt sind, spricht man von einer *heterogenen Population*. Liegen die Individuen hingegen im Lösungsraum relativ nahe zusammen oder sind sie sogar identisch, handelt es sich um eine *homogene Population*. Heterogene Startpopulationen werden üblicherweise erzeugt, indem die Parameterwerte der Individuen mit Zufallszahlen aus den zulässigen Wertebereichen initialisiert werden. Eine solche zufällige Parameterbelegung bietet sich jedoch für die gegebene Problemstellung nicht an, da bereits ein konkretes Segmentierungsergebnis zugrunde liegt, auf welches die Anwendenden gegebenenfalls sogar Einfluss (vgl. Abschnitt 5.3) genommen haben. Das so entstandene Segmentierungsergebnis dient als Ausgangspunkt für die Optimierung, was sich mit einer heterogenen Startpopulation so nicht realisieren ließe.

Bei der vorliegenden Problemstellung wird eine Startpopulation gewählt, die μ viele identische Individuen enthält: Die Werte der Objektvariablen werden direkt aus den Parametern abgeleitet, die der Initialsegmentierung zugrunde lagen. Während solche homogenen Startpopulationen für Genetische Algorithmen problematisch sind, ist dies

[7] In dieser Dissertation wird davon ausgegangen, dass die Dreiecke als Liste vorliegen und über einen Index adressiert werden können.

bei Evolutionsstrategien nicht der Fall, da bei diesen – aufgrund der weitaus größeren Bedeutung der Mutation – die genetische Vielfalt nicht schon in der Startpopulation vorhanden sein muss [Ort02].

7.2.3 Reproduktion und Selektion

Rekombination und Mutation werden gelegentlich unter dem Begriff *Reproduktion* zusammengefasst. Gemeinsam bewirken sie die Erzeugung von Offspring-Individuen, welche sich im Normalfall sowohl voneinander unterscheiden als auch von den Parent-Individuen.

Rekombination

Bäck gibt in [Bäc96] sieben unterschiedliche Rekombinationsmechanismen an. Er geht dabei von einer geschlechtlichen Rekombination aus, bei welcher der lokale Rekombinationsoperator $r' : I^2 \to I$ aus zwei zufällig ausgewählten Individuen S und T ein neues erzeugt, wobei I den Raum der Individuen bezeichnet[8]. Aus den Rekombinationsmechanismen resultieren unterschiedliche Ausprägungen des Operators r'. Diese sind detailliert in [Bäc96] aufgeführt. Für die vorliegende Arbeit sind folgende relevant:

- *Diskrete Rekombination (Operator r'_d):* Für jedes Gen wird per Zufall ermittelt, ob es aus dem Individuum S oder aus T übernommen wird. Die Wahrscheinlichkeit ist für beide Fälle exakt gleich groß.
- *Intermediäre Rekombination (Operator r'_i):* Der Wert eines Gens ergibt sich durch Mittelung der entsprechenden Gene der Individuen S und T.

Bei den beiden zufällig ausgewählten Individuen S und T kann es sich durchaus auch um dasselbe Individuum handeln. Die Rekombination verursacht dann allerdings keinerlei genetische Veränderungen, was aber auch kein Problem darstellt. Die Wahrscheinlichkeit für eine solche Situation liegt bei $\frac{1}{\mu}$.

Prinzipiell kann für jedes Chromosom der Individuen (Objektvariablen, Schrittweiten und gegebenenfalls die Rotationswinkel) ein eigener Rekombinationsoperator verwendet werden [Bäc96]. Dieser Ansatz ist in der vorliegenden Arbeit aufgegriffen und derart erweitert worden, dass auch für die einzelnen Objektvariablen unterschiedliche Operatoren zum Einsatz kommen. Der Rekombinationsoperator r' für den Vektor der Objektvariablen setzt sich in dieser Arbeit aus einem Operator für die Seed-Punkte r'_s sowie mit r'_ζ, r'_η und r'_ν aus jeweils einem Operator für die Parameter ζ, η und ν zusammen. Für die einzelnen Operatoren können verschiedene Rekombinationsstrategien gewählt

[8] Wie schon in Unterabschnitt 7.1.1 werden auch weiterhin lokale Operatoren entsprechend der Notation aus [BS93] mit einem Strich versehen.

werden. Dabei tendiert eine diskrete Rekombination dazu, die vorhandene Heterogenität zu erhalten, während eine intermediäre Rekombination heterogenitätsmindernd wirkt [Nis94, S. 141].

Die Parameter ζ und η sind reellwertig, so dass r'_ζ und r'_η beim EvOpSeg-Verfahren als intermediäre Operatoren r'_i realisiert werden. Der Wertebereich von ν ist hingegen durch die Menge $\{0, 1\}$ gegeben. Eine intermediäre Rekombination kann somit zu Werten führen, die außerhalb dieser Menge liegen. Deswegen wird für die Rekombination von ν ein diskreter Rekombinationsoperator $r'_\nu = r'_d$ verwendet.

Für die Seed-Positionen scheidet eine intermediäre Rekombination aus, da eine Mittlung der entsprechenden Positions-Indizes im Allgemeinen offensichtlich zu keinem sinnvollen Ergebnis führt. Dies liegt daran, dass die Ordnung der Indizes nicht mit der Anordnung der Dreiecke übereinstimmt. Deswegen werden die Seed-Punkte diskret rekombiniert. Allerdings muss der Rekombinationsoperator so beschaffen sein, dass Patch-Gruppen in der Regel nicht in unzusammenhängende Teile zerfallen. Aus diesem Grund scheint es sinnvoll zu sein, nicht nur einzelne Seed-Punkte, sondern ganze Patch-Gruppen bei der Rekombination zu betrachten: Für jede Patch-Gruppe werden sämtliche zugehörigen Seed-Punkte entweder aus dem Individuum S oder aus dem Individuum T übernommen. Dies minimiert die Wahrscheinlichkeit, dass die Patch-Gruppe im rekombinierten Individuum nicht zusammenhängend ist. Darüber hinaus wird so auch das Auftreten von „doppelten Seed-Punkten“ ausgeschlossen, also von Situationen, in denen mehrere Seed-Punkte der Patch-Gruppe auf demselben Dreieck des Oberflächennetzes liegen.

Um die an Patch-Gruppen orientierte Rekombination der Seed-Punkte formal zu beschreiben, wird der Operator r'_s in m Teiloperatoren $\hat{r}'_l$ mit $l \in \{1, \ldots, m\}$ aufgeteilt, die unabhängig voneinander sind und parallel angewendet werden können. Dabei bezeichnet m wie schon in Abschnitt 7.2.1 die Anzahl der Patch-Gruppen. Der Rekombinationsoperator für die l-te Patch-Gruppe ist definiert durch

$$\hat{r}'_l(\vec{s}_l^{(S)}, \vec{s}_l^{(T)}) := \begin{cases} \vec{s}_l^{(S)} & \text{, falls } \chi_r = 0 \\ \vec{s}_l^{(T)} & \text{, falls } \chi_r = 1, \end{cases} \tag{7.6}$$

wobei χ_r eine gleichverteilte Zufallsvariable mit dem Wertebereich $\{0, 1\}$ ist. Die hochgestellten Indizes geben an, ob die Seeds einer Patch-Gruppe von Individuum S oder von T betrachtet werden. Die Teiloperatoren $\hat{r}'_l$ bewirken eine diskrete Rekombination der Parameter. Ähnlich wie beim Operator r'_d liefern sie $\vec{s}_l^{(S)}$ und $\vec{s}_l^{(T)}$ mit derselben Wahrscheinlichkeit.

Als Rekombinationsoperator für die Mutationsschrittweite σ wird ein intermediärer Operator gewählt. Der neue Wert für σ entspricht dem arithmetischen Mittel der σ-Werte beider Parent-Individuen.

Mutation

Ein Individuum wird mutiert, indem jede Komponente seines Parametervektors einzeln einer Mutation unterzogen wird. Diese Mutation muss allerdings nicht zwangsläufig zu

einer Änderung des Parameterwertes führen: Die reellwertigen Parameter (ζ, η sowie die Seed-Gewichte $\gamma_{i,j}$) werden mutiert, indem jeweils eine normalverteilte Zufallszahl $\chi \in \mathbb{R}$ mit dem Erwartungswert 0 und einer Standardabweichung σ_ζ, σ_η bzw. σ_γ hinzu addiert wird. Dabei ist χ für jeden Parameter neu zu ermitteln.

Die Mutation der Seed-Positionen kann nicht wie im Fall von reellwertigen Parametern durch einfaches Addieren einer Zufallszahl erfolgen. Das liegt daran, dass die Seed-Positionen als Indizes der entsprechenden Dreiecksflächen kodiert sind. Auch bei den Seed-Positionen soll die Mutation häufiger zu kleinen Änderungen führen als zu großen Änderungen. Unter einer Mutation der Position eines Seeds versteht man dabei das Verschieben des Seeds auf der Oberfläche. Der Mutationsoperator hat also so beschaffen zu sein, dass die Seeds eher lokal begrenzt verschoben werden, als über eine größeren Distanz. Da die Seed-Punkte lediglich diskret, nämlich auf den Schwerpunkten von Dreiecksflächen, angeordnet sein können, muss auch der Mutationsoperator diskret realisiert sein. Im Folgenden werden zwei Varianten für die Mutation der Seed-Positionen beschrieben.

- **Variante 1: Freie Mutation der Seed-Positionen**
  Es werden zufällige Werte für die Distanz (in Anzahl von Dreiecken) und Richtung bestimmt, um die bzw. in die ein Seed-Punkt $\mathbf{p}_{seed}$ zu verschieben ist. Die Distanz ist durch den Absolutwert einer normalverteilten Zufallszahl χ' mit Erwartungswert 0 und Standardabweichung σ_p gegeben. Dann werden alle Dreiecke ermittelt, die von $\mathbf{p}_{seed}$ den dreieckbasierten Abstand $round(|\chi'|)$ haben und die keinen Seed-Punkt enthalten. Von diesen wird eins zufällig als neues Seed-Dreieck gewählt, dessen Index den des alten Seed-Dreiecks im Individuum ersetzt. Wenn kein solches Dreieck existiert, bleibt die Position von $\mathbf{p}_{seed}$ unverändert.

- **Variante 2: Gebundene Mutation der Seed-Positionen**
  Bei dieser Variante darf jeder Satelliten-Seed höchstens ω viele Dreiecke von seinem Initial-Seed entfernt liegen, so dass man die Satelliten-Seeds in gewissem Sinne als an ihren Initial-Seed „angebunden“ ansehen kann. Im Gegensatz zur freien Mutation der Seed-Positionen ist nun die Reihenfolge, in der die Seeds betrachtet werden, sehrwohl von Bedeutung (vgl. Abbildung 7.5). Zunächst werden alle Initial-Seeds entsprechend der ersten Variante verschoben, d.h. es findet eine freie Mutation der Initial-Seed-Positionen statt. Anschließend werden alle Satelliten-Seeds verschoben. Auch hierbei ergibt sich die Distanz, um die ein solcher Seed $\mathbf{p}_{sat}$ zu verschieben ist, durch den Absolutwert einer normalverteilten Zufallszahl χ' mit Erwartungswert 0 und Standardabweichung σ_p. Von allen Dreiecken, die von $\mathbf{p}_{sat}$ den dreieckbasierten Abstand $round(|\chi'|)$ haben, keinen Seed-Punkt enthalten und zudem nicht weiter als ω Dreiecke von dem zugehörigen, bereits verschobenen Initial-Seed $\mathbf{p}_{initial}$ entfernt liegen, wird ein zufälliges gewählt. Existiert kein solches Dreieck, wird $\mathbf{p}_{sat}$ derart verschoben, dass er von $\mathbf{p}_{initial}$ maximal den dreieckbasierten Abstand ω hat und zugleich möglichst nah an seiner vorherigen Position liegt.

Wird eine gebundene Mutation mit $\omega = \infty$ durchgeführt, entspricht dies einer freien Mutation. Bei dieser können zusammengehörige Seeds durch die Mutation zu weit von-

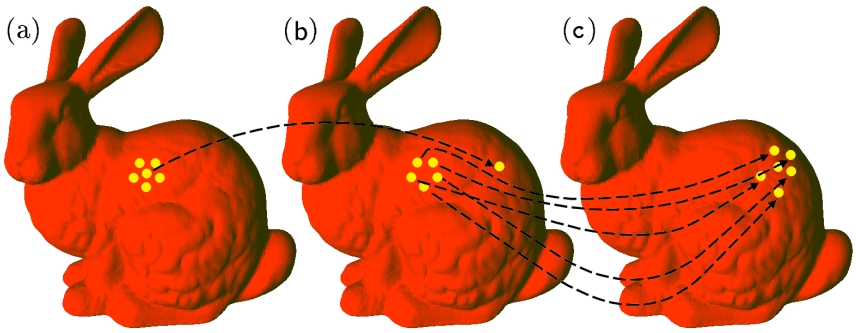


Abbildung 7.5: *Veranschaulichung der gebundenen Mutation: Von allen zu mutierenden Seed-Positionen (a) werden im ersten Schritt nur die Initial-Seeds berücksichtigt (b). Im zweiten Schritt werden dann die Positionen aller Satelliten-Seeds so mutiert, dass die Satelliten-Seeds anschließend innerhalb einer vorgegebenen Nachbarschaft um den zugehörigen Initial-Seed liegen (c).*

einander weg verschoben werden, so dass daraus ein Patch resultiert, welches aus mehreren nicht-zusammenhängenden Teile besteht. Mit einem solchen Zerfallen der Patches ist hingegen bei einer „echten“ gebundenen Mutation mit einem sinnvoll gewählten ω im Normalfall nicht zu rechnen.

Ähnlich wie bei Genetischen Algorithmen wird für den binären Parameter ν, der angibt, ob die winkelbasierte oder die merkmalbasierte Distanzfunktion zu verwenden ist, eine Mutationswahrscheinlichkeit $p_m \in [0, 1]$ verwendet (vgl. [Bäc96, S. 113]). Der mutierte Wert von ν ergibt sich unter Berücksichtigung einer gleichverteilten Zufallsvariablen $\chi'' \in [0, 1]$ nach:

$$\nu_{neu} := \begin{cases} \nu & \text{, falls } \chi'' > p_m \\ 1 - \nu & \text{, falls } \chi'' \leq p_m \end{cases} \tag{7.7}$$

Wenn der Wert von χ'' kleiner ist als die Mutationswahrscheinlichkeit, wird also der Wert von ν invertiert. Während die Mutationswahrscheinlichkeit bei Genetischen Algorithmen eher klein – Bäck gibt in [Bäc96] Werte zwischen 0.001 und 0.01 an – sein sollte, ist im vorliegenden Fall sicherlich ein größerer Wert zu wählen, da der Mutation bei Evolutionsstrategien eine wesentlich stärkere Bedeutung zukommt.

Anders als in [EM12] wird im Folgenden eine Selbstadaption der Mutationsschrittweite verwendet, bei der die Mutationsschrittweite als zusätzlicher zu optimierender Strategieparameter genommen wird. Ursem stellt in [Urs03, S. 59] heraus, dass die Selbstadaption im Allgemeinen eine entsprechend große Anzahl von Generationen erfordert. Deswegen ist hier zunächst die von Hebbel als *einfache Mutation* [Heb09, S. 38] bezeichnete, weniger komplexe Variante gewählt worden, bei der – wie schon in Abschnitt 7.1.1 erläutert – ein globales σ für alle Objektvariablen vorliegt. Aus diesem werden die oben erwähnten Standardabweichungen σ_ζ, σ_η, σ_γ sowie σ_p abgeleitet. Die genaue Konstruktion wird in Abschnitt 8.2 beschrieben.

Selektion

Der in dieser Arbeit verwendete Selektionsoperator entspricht dem einer (μ, κ, λ)-Evolutionsstrategie. Die Wahl ist auf diesen Operator gefallen, da sich mit ihm abhängig von κ auch eine Plus-Strategie oder eine Komma-Strategie realisieren lässt. Betrachtet wird die Menge aller Parent- und Offspring-Individuen, deren Alter geringer ist als die maximale Lebenserwartung κ (vgl. Abschnitt 7.1.1). Aus dieser Menge werden die μ Individuen mit den besten Fitnesswerten als Parents der neuen Generation selektiert. Solange die Fitnesswerte paarweise verschieden sind, erfolgt die Selektion vollkommen deterministisch.

7.2.4 Definition einer geeigneten Zielfunktion

Im Gegensatz zu anderen Evolutionären Algorithmen ist bei Evolutionsstrategien stets der Fitnesswert eines Individuums durch den Zielfunktionswert des Objektvariablenvektors gegeben [Bäc96]. Deswegen werden im Folgenden die Begriffe „Zielfunktion“ und „Fitnessfunktion“ synonym verwendet, sofern nicht anders angegeben. Die evolutionäre Optimierung der Patches ist in dieser Dissertation als Minimierungsproblem formuliert, d.h. ein kleiner Fitnesswert soll darauf hindeuten, dass die durch das Individuum implizit gegebenen Patches als „gut“ anzusehen sind. Um zu verhindern, dass so genannte *ungültige Individuen* als neue Eltern-Individuen selektiert werden, müssen sie also einen Fitnesswert erhalten, der größer ist als die möglichen Fitnesswerte der gültigen Individuen. Entsprechend der nachfolgend beschriebenen Konstruktion der Zielfunktion reicht es aus, einen Wert zu nehmen, der größer als 1 ist. Als *ungültige Individuen* werden Individuen bezeichnet, die zu ungültigen Segmentierungen korrespondieren. Eine *ungültige Segmentierung* liegt vor, wenn mindestens ein Patch oder eine Patch-Gruppe in mehrere Teile zerfällt, die nicht zusammenhängen. Jede Patch-Gruppe wird im weiteren Verlauf als *Segment* angesehen.

Wie schon anhand der Betrachtung der ungültigen Individuen zu erkennen ist, kann der Fitnesswert eines Individuums bzw. der Zielfunktionswert nicht direkt aus den Parameterbelegungen abgeleitet werden. Stattdessen ergibt er sich aus der Qualität der korrespondierenden Segmentierung. Für einen Vektor von Objektvariablen $\vec{x} = (\vec{s}, \zeta, \eta, \nu)$ ist die Zielfunktion f als Hintereinanderausführung von zwei Funktionen definiert:

$$f(\vec{x}) := h \circ g(\vec{x}). \tag{7.8}$$

Dabei führt $g(\vec{x})$ zu einer Segmentierung entsprechend dem Ergebnis des Algorithmus zur Patch-Berechnung aus Listing 5.1. An dieser Stelle sei aus Gründen der Vollständigkeit darauf hingewiesen, dass der Algorithmus nicht als Funktion im formalen Sinne angesehen werden kann. Das liegt daran, dass bei zwei Seed-Punkten mit identischen Seed-Gewichten keine explizite Berechnungsreihenfolge definiert ist. Zwei unterschiedliche Reihenfolgen können durchaus zu leicht anderen Segmentierungen führen. Dieser Unterschied ist jedoch so gering, dass er in der Praxis zu vernachlässigen ist. Alternativ könnte auch eine Reihenfolge explizit festgelegt werden.

Angewendet auf eine gültige Segmentierung Γ liefert die Funktion h den Fitnesswert des Individuums, durch welches Γ induziert worden ist. Entsprechend der Erkenntnisse über die menschliche Wahrnehmung, die Hoffmann und Richards in [HR84] vorstellen, sind Segmente erwünscht, die größtenteils von konkaven Gebieten umgeben sind. Andererseits sind die Ränder der Komponenten, die von Menschen intuitiv als solche erkannt werden, gewöhnlicherweise nicht oder zumindest nicht stark „ausgefranst“. Neben dem genauen Verlauf unterscheiden sich ausgefranste Ränder von nicht-ausgefransten auch insofern, als dass sie länger sind. Demnach sind eher kurze Segmenträndern erwünscht. Diese beiden Annahmen, dass einerseits kurze Segmenträndern gesucht werden, die aber andererseits auch innerhalb von konkaven Objektregionen liegen, definieren jeweils eine eigene Zielfunktion, so dass ein mehrkriterielles Optimierungsproblem vorliegt. Gelöst wird dieses Optimierungsproblem durch Rückführung auf ein einkriterielles Optimierungsproblem mit Hilfe der in Abschnitt 7.1.4 erläuterten Weighted-Sum-Methode. Mit den Gewichtungsfaktoren $(1-\Lambda)$ respektive Λ wird dann die Funktion h als

$$h(\Gamma) := \begin{cases} (1-\Lambda)\cdot h_{concave}(\Gamma) + \Lambda \cdot h_{length}(\Gamma); \text{mit } \Lambda \in [0,1] & \text{falls } \Gamma \text{ gültig ist,} \\ \infty & \text{ansonsten,} \end{cases} \tag{7.9}$$

definiert, wobei h_{length} und $h_{concave}$ die beiden Zielfunktionen des mehrkriteriellen Optimierungsproblems sind. Die Funktion $h_{concave}$ gibt an, wie konkav die Bereiche des Objekts sind, in denen die Segmenträndern verlaufen. Dabei werden allerdings nur die Ränder von Segmenten betrachtet, und nicht etwa Ränder zwischen mehreren Patches, die eine Patch-Gruppe bilden und damit zum selben Segment gehören. Da der Algorithmus zur Patch-Berechnung keine Netzelemente wie Dreiecke oder Kanten weiter unterteilt, sind die Segmenträndern in Form von Polygonzügen gegeben, deren Elemente Kanten des Dreiecksnetzes sind. Die Konkavitätsinformationen sind über die dihedralen Winkel an den Kanten gegeben. Eine lange Kante soll einen größeren Einfluss auf $h_{concave}$ haben als eine kürzere. Hingegen soll der Anzahl der Kanten, die den Rand des Segments bilden, keine Bedeutung zukommen. $h_{concave}$ wird damit wie folgt definiert:

$$h_{concave}(\Gamma) := \underbrace{\frac{1}{\sum_{i=1}^{|E|} \frac{1}{l(e_i)}}}_{\text{Ⓐ}} \cdot \sum_{j=1}^{|E|} \left(\frac{1}{1+\max(\alpha_j, 0)} \cdot \frac{1}{l(e_j)} \right). \tag{7.10}$$

Dabei sei E die Liste aller Kanten des Dreiecksnetzes, die zu den Segmenträndern der Segmentierung Γ gehören. Da eine solche Randkante eines Segments zugleich auch Randkante eines anderen Segments ist, wird sie in dieser Arbeit auch *Grenzkante* genannt. Die i-te Grenzkante wird mit e_i und deren Länge mit $l(e_i)$ bezeichnet. α_i bezeichnet den vorzeichenbehafteten Winkel zwischen den Normalenvektoren der beiden Dreiecksfacetten, die an e_i angrenzen. Dieser Winkel ist positiv, wenn das Objekt konkav an der Kante e_i ist. Der mit Ⓐ gekennzeichnete erste Faktor in Gleichung (7.10) skaliert das Ergebnis, so dass stets $f_{concave} \in [0,1]$ gilt. Je stärker die Konkavität der Segment-Grenzen von Γ ist, umso kleiner ist der Funktionswert von $f_{concave}$.

Mit der Funktion

$$h_{length}(\Gamma) := \frac{1}{\sum_{i=1}^{|E_{mesh}|} l(e'_i)} \cdot \sum_{j=1}^{|E|} l(e_j) \tag{7.11}$$

wird bestimmt, wie ausgefranst die Segmenträndersind, indem dies auf die Länge der Ränder zurückgeführt wird. Je glatter die Segmenträndersind, umso kleiner ist der Funktionswert. Auch hierbei wird die Liste aller Kanten, die zu Segmenträndern gehören, mit E bezeichnet; die Liste aller Kanten des Dreiecksnetzes ist als E_{mesh} bezeichnet. Während e_j das j-te Element von E angibt, bezeichnet e'_i die i-te Kante von E_{mesh}. Die Verknüpfung von $h_{concave}$ und h_{length} in Gleichung (7.9) ist als Konvexkombination realisiert. Dadurch ist sichergestellt, dass für gültige Segmentierungen der Wertebereich von h eine Teilmenge des Intervalls $[0, 1]$ ist. Da sowohl $h_{concave}$ als auch h_{length} Minimierungsprobleme beschreiben, ist ein minimaler Wert der Zielfunktion f zu ermitteln.

7.3 Zweistufige Patch-Optimierung

Die evolutionäre Patch-Optimierung kann durch Seed-Verschiebungen schlecht gewählte Initial-Seed-Positionen automatisch „korrigieren“. Sofern keine Satelliten-Seeds vorhanden sind, geht dies gewöhnlicherweise damit einher, dass sich die Lage von mindestens einem Patch signifikant verändert. Mit Satelliten-Seeds können zwar die Ränder eines Patches recht gut beeinflusst werden, sie erschweren allerdings das Verschieben eines kompletten Patches durch die im vorherigen Abschnitt beschriebenen Variationsoperatoren, welches wiederum notwendig sein kann, damit semantisch bedeutungsvolle Segmente entstehen.

Abhilfe kann eine *zweistufige Patch-Optimierung* schaffen, bei der zunächst versucht wird, die Patches an geeigneten Orten zu platzieren, so dass sie sich nicht allzu sehr über Komponentengrenzen hinweg erstrecken. Erst danach wird der Fokus auf die Optimierung der Patch-Grenzen gelegt. Die beiden Stufen sehen dann wie folgt aus:

Erste Stufe: Ausschließlich unter Nutzung von Initial-Seeds wird solange eine evolutionäre Patch-Optimierung durchgeführt, bis ein Abbruchkriterium greift. Bei dem Abbruchkriterium kann es sich beispielsweise um eine vorgegebene Zeitdauer oder um eine vorgegebene Anzahl von Generationen handeln. Die Optimierung kann aber auch abgebrochen werden, wenn über eine gewisse Anzahl von Generationen hinweg keine nennenswerte Verbesserung der Fitnesswerte stattgefunden hat.

Zweite Stufe: In der zweiten Stufe soll im Wesentlichen der Verlauf der Patch-Grenzen adaptiert werden, aber nicht unbedingt eine nennenswerte Verschiebung der Patches erfolgen. Dazu werden ausgehend von der durch die Optimierung der ersten

Stufe erzielten Situation für jeden Initial-Seed nun Satelliten-Seeds generiert. Anschließend findet erneut eine evolutionäre Patch-Optimierung statt, bei der aber zusätzlich auch die neu erzeugten Satelliten-Seeds zu berücksichtigen sind.

Sind die Initial-Seeds, die als Basis für die erste Stufe genommen werden, manuell auf dem Modell platziert worden, sollte man davon ausgehen können, dass dies mit entsprechender Sorgfalt geschehen ist und kein Initial-Seed deutlich zu dicht an Komponentengrenzen liegt. Es ist fraglich, ob die mehrstufige Optimierung in einer solchen Situation einen Vorteil bietet.

7.4 Parallelisierung

Evolutionäre Algorithmen benötigen bei zahlreichen Problemstellungen einen enormen Rechenaufwand. Sie beinhalten jedoch auch ein großes Parallelisierungspotential, so dass sich die benötigte Optimierungszeit durch eine parallele Verarbeitung auf mehreren unabhängigen Recheneinheiten deutlich reduzieren lässt. Dabei ist zwischen grundsätzlich verschiedenen Parallelisierungsarten zu unterscheiden:

1. **Parallelisierung durch Verwendung mehrerer Populationen:** Es gibt unterschiedliche Ansätze, bei denen mehrere Populationen parallel verwendet werden. Eines von denen ist das *Insel-Modell* [GS90, Spr99, WMR01], bei dem nur Individuen rekombiniert werden können, die zur selben Population gehören[9]. Zu bestimmten Zeitpunkten, die durch ein *Migrationsintervall* [NORS11] definiert sind, können einzelne Individuen von einer Population zu einer anderen migrieren. Ist das Migrationsintervall so gewählt, dass während des Optimierungsprozesses keine Migration vorkommt, liegen lediglich „normale“, parallel zueinander erfolgende Optimierungsläufe vor. Während beim Insel-Modell für die Migration keine weiteren Einschränkungen gelten, existiert beim *Stepping-Stone-Modell* eine Nachbarschaftsbeziehung zwischen den Populationen [GS90, Spr99, WMR01]. Migriert werden darf dort nur zwischen benachbarten Populationen.

2. **Parallelisierung durch zellulare Aufteilung:** Eine derartige Parallelisierung, die auch als *fine-grained Parallelisierung* bekannt ist, beschreiben Lim et al. [LOJ+07] im Zusammenhang mit Genetischen Algorithmen. Diese Art der Parallelisierung ist hervorragend für Prozessor-Systeme geeignet, die eine Grid-Struktur aufweisen. Die Prozessoren sind klar strukturiert angeordnet, so dass jeder von ihnen feste Nachbarn besitzt. Betrachtet wird eine einzige Population, deren Individuen jeweils einem Prozessor zugeordnet sind. Zur Rekombination und Selektion werden nur die Individuen solcher Prozessoren betrachtet, die zu dem aktuellen benachbart sind.

3. **Parallelisierung durch Master-Slave-Ansätze:** Auch bei dieser Variante wird lediglich eine einzige Population verwendet. Ziel ist es, die Operatoren des

[9] Gelegentlich wird auch von einer *coarse-grained Parallelisierung* gesprochen.

Evolutionären Algorithmus zu parallelisieren. Beispielsweise erfolgt die Fitnessberechnung für mehrere Offspring-Individuen zur selben Zeit auf unterschiedlichen Prozessoren. Ein Master-Prozess steuert den globalen Ablauf und verteilt Aufgaben, die parallel bearbeitet werden können, an Slave-Prozesse auf unterschiedlichen Prozessoren bzw. Prozessor-Kernen [CP97, GB06, LOJ+07]. Im Extremfall können sogar einzelne lokale Operatoren parallelisiert werden, so dass beispielsweise die einzelnen Gene parallel betrachtet werden. Lim et al. bezeichnen dies als *2nd Level Parallelism* [LOJ+07].

Eine Parallelisierung der Operatoren kann durchaus auch bei Verwendung mehrerer Populationen, also beispielsweise beim Insel-Modell, angewendet werden. Nach [CP97] konvergieren Insel-Modelle zumindest bei Genetischen Algorithmen schneller als bei den „seriellen“ Formen. Allerdings könne die Qualität der Lösung dabei auch schlechter sein. Insel-Modell, Stepping-Stone-Modell sowie die zellulare Aufteilung stellen im Grunde eine *Parallelisierung des Modells* dar, während Parallelisierung der Operatoren als eine *Parallelisierung der Implementierung* angesehen werden kann. Werden Modelle parallelisiert, bedeutet dies, dass ein abgewandelter Optimierungsalgorithmus verwendet wird, der durchaus zu anderen Ergebnissen führen kann. In der vorliegenden Arbeit soll jedoch nur die Rechenzeit reduziert werden, ohne dass der zugrunde liegende Algorithmus verändert wird. Deswegen wird im Folgenden ausschließlich auf Parallelisierungsmöglichkeiten der Operatoren eingegangen.

7.4.1 Möglichkeiten der Parallelisierung von Operatoren

Aktuelle Rechner verfügen über mehrere Prozessoren oder unabhängige Prozessor-Kerne, die bei Problemstellungen mit einer hohen Parallelisierbarkeit zu einer beachtlichen Reduktion der benötigten Rechenzeit führen können. Zukünftig ist eine weitere Zunahme der Kerne pro Prozessor sowie eine weitere Verbreitung von Multiprozessor-Systemen zu erwarten. Der maximal mögliche Speedup bei einer parallelen Umsetzung der Operatoren ist durch die Anzahl der Prozessoren bzw. Prozessor-Kerne begrenzt. Dies gilt insbesondere dann, wenn weniger als λ Prozessoren bzw. Prozessor-Kerne vorhanden sind. Andernfalls wirkt sich die Anzahl λ der Offspring-Individuen limitierend auf den maximal möglichen Speedup aus, was in Abschnitt 8.7 erläutert wird. Der in der Praxis erzielbare Speedup kann hingegen deutlich niedriger ausfallen, da beispielsweise der benötigte Mehraufwand zur Verwaltung der Daten ebenfalls limitierend wirkt (vgl. [Amd67]).

Im Folgenden wird kurz analysiert, wie weit sich eine Evolutionsstrategie parallelisieren lässt, wenn nur eine einzige Population vorliegt. Dabei wird auf die einzelnen Schritte der Evolutionsstrategie separat eingegangen. Nicht alle Schritte bieten sich für eine Parallelisierung an (vgl. beispielsweise [GKK04]). Der größte Nutzen ist im vorliegenden Fall, ähnlich wie bei dem in [PADC05] geschilderten, aus der Parallelisierung der Fitnessberechnung zu erwarten, da die Berechnung der Fitness in der Regel den größten Teil der insgesamt benötigten Rechenzeit beansprucht (vgl. auch [DLJD00, S. 292]).

Parallelisierungsmöglichkeiten bei der Erzeugung der Startpopulation

Im Gegensatz zu den anderen Schritten, die für jede Generation durchzuführen sind, erfolgt die Erzeugung der Startpopulation nur einmal am Anfang. Bei der für die evolutionäre Patch-Optimierung gewählten Evolutionsstrategie besteht sie aus der Übernahme der Parameter der Initialsegmentierung. Sämtliche Individuen der Startpopulation sind identisch, so dass lediglich Zeit für das Kopieren der Daten, nicht jedoch für eine Berechnung der Individuen benötigt wird. Die für das Kopieren der Parameterwerte anfallende Zeit ist jedoch zu vernachlässigen, zumal die Erzeugung der Startpopulation nicht innerhalb der Optimierungsiteration geschieht.

Parallelisierungsmöglichkeiten bei der Rekombination

Bei der geschlechtlichen Rekombination werden jeweils zwei ausgewählte Individuen betrachtet, die hier als *Paar* bezeichnet werden. Die Durchführung einer Rekombination für ein konkretes Paar hat keinen Einfluss auf andere Paare. Folglich können die Paare parallel zueinander rekombiniert werden.

Parallelisierungsmöglichkeiten bei der Mutation

Die einzelnen Individuen werden unabhängig voneinander mutiert, so dass sie im Mutationsschritt parallel betrachtet werden können. Da es sich um λ viele zu mutierende Individuen handelt, kann auf diese Weise maximal ein Speedup von λ erzielt werden. Dies gilt selbst dann, wenn mehr als λ Prozessoren bzw. Kerne zur Verfügung stehen. Eine weitere Steigerung des Speedups lässt sich erzielen, wenn bei einer genügend großen Anzahl an Prozessor-Kernen auch die einzelnen Gene eines jeden Indiviuums unabhängig voneinander mutiert werden.

Parallelisierungsmöglichkeiten bei der Berechnung der Fitnesswerte

Die Bewertung der Individuen in Form der Berechnung ihrer Fitnesswerte verursacht bei der evolutionären Patch-Optimierung einen wesentlichen Teil der insgesamt anfallenden Rechenzeit. Von einer Parallelisierung dieses Schritts ist somit der größte Nutzen zu erwarten. Wie bei der Mutation erfolgt auch die Fitnessbewertung der einzelnen Individuen vollkommen unabhängig voneinander, so dass sie problemlos parallel durchgeführt werden kann. Hingegen können weder die einzelnen Gene unabhängig voneinander betrachtet werden, noch die Patch-Berechnung für die einzelnen Seed-Punkte parallel erfolgen. Die Begründung ist die gleiche wie die in Abschnitt 5.2.3 angegebene: Bei einer parallelen Patch-Berechnung könnten aufgrund unterschiedlicher Seed-Gewichte durchaus nicht-zusammenhängende Patches entstehen.

Parallelisierungsmöglichkeiten bei der Selektion

Der in dieser Arbeit verwendete Selektionsoperator arbeitet deterministisch, sofern alle Individuen paarweise unterschiedliche Fitnesswerte besitzen. Es werden stets die besten μ Individuen aus der Menge aller für die Selektion in Frage kommenden genommen. Demzufolge sind die Individuen zu sortieren, was impliziert, dass sie nicht unabhängig voneinander betrachtet werden können. Somit ist auch kein relevantes Parallelisierungspotential vorhanden.

7.4.2 Parallelisierung beim EvOpSeg-Verfahren

Beim EvOpSeg-Verfahren soll eine deutliche Reduktion der für die Optimierung benötigten Rechenzeit durch Parallelisierung der Operatoren erzielt werden. Eine parallele Erzeugung der Start-Population hilft dabei nicht, zumal diese Population im vorliegenden Fall stets aus identischen Individuen besteht. Wie beschrieben bietet auch der Selektionsoperator kein wirkliches Potential für eine Parallelisierung. Der Mutationsoperator ist hingegen parallel implementiert worden. Dies hatte allerdings nur einen vernachlässigbaren Einfluss auf die insgesamt benötigte Rechenzeit.

Anders als bei den anderen Operatoren wirkt sich die Modellgröße direkt auf die für die Fitnessberechnung benötigte Zeit aus. Die Fitnessberechnung beinhaltet die Ermittlung aller Patches, bei der Dijkstras Algorithmus zur Bestimmung kürzester Wege [Dij59] zum Einsatz kommt. Je nach Modell wird allein für die Patch-Berechnung eine oder mehrere Sekunden benötigt. Hinzu kommt noch die Zeit zur Bestimmung der eigentlichen Fitnesswerte, also die Berechnung von $h_{concave}$ und h_{length} nach den Gleichungen (7.10) und (7.11). In diese Berechnung fließen alle Kanten ein, die zu Segment-Rändern gehören. Je feiner das Modell ist, umso mehr solcher Kanten liegen vor. Damit verursacht die Fitnessberechnung im gesamten Optimierungsablauf den mit Abstand größten Aufwand. Verglichen mit ihr ist die Rechenzeit, die durch Variations- und Selektionsoperator anfällt, zu vernachlässigen. Folglich ist der größte Speedup durch eine Parallelisierung der Fitnessberechnung zu erwarten. Dies ist beim EvOpSeg-Verfahren nach dem Master-Slave-Ansatz realisiert worden: Jede Recheneinheit berechnet die Fitness eines Individuums. Wenn mehr Individuen vorhanden sind, als es Recheneinheiten gibt, werden noch nicht ausgewertete Individuen von einem Master-Prozess an freie Recheneinheiten verteilt.

7.5 Ergebnisse

Zur Demonstration der Auswirkung der Optimierung ist hier exemplarisch das Modell einer Hand herangezogen worden. Die Ausgangssituation in Form einer Initialsegmentierung ist in Abbildung 7.6(a) dargestellt. Wie bereits in Abschnitt 5.4 erwähnt, sind weder das grüne Patch (Zeigefinger) noch das gelbe Patch (Mittelfinger) optimal.

Ausgehend von dieser Situation sind zwei Optimierungsläufe durchgeführt worden: eine (μ, κ, λ)-ES und eine $(\mu + \lambda)$-ES. Die Optimierung ist jeweils mit 20 Parent- und 100 Offspring-Individuen über 150 Generationen erfolgt. Die Ergebnisse der beiden Optimierungen sind in Abbildung 7.6(b) bzw. 7.6(c) zu sehen. Sie sind deutlich besser als die Initialsegmentierung und entsprechen den Erwartungen an eine gute Segmentierung. Die zugehörigen Fitnessverläufe für die besten Individuen sind ebenfalls Abbildung 7.6 zu entnehmen. Hierzu sei angemerkt, dass die Fitnesswerte selbst bei einer optimalen Segmentierung nach Konstruktion der Fitnessfunktion stets deutlich größer als 0 sind.

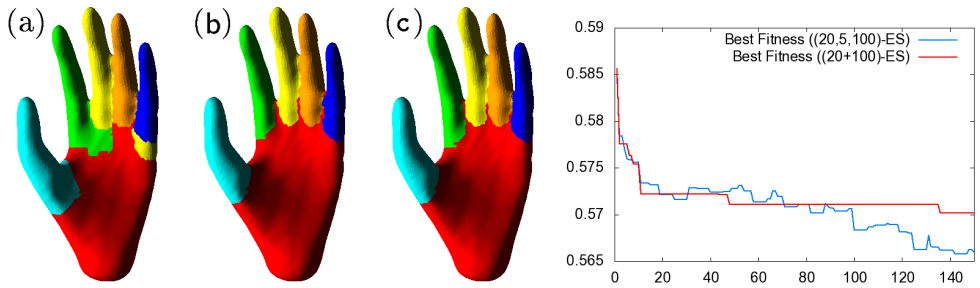


Abbildung 7.6: *Die Initialsegmentierung (a) wurde mit einer (μ, κ, λ)-ES (b) und einer $(\mu+\lambda)$-ES (c) optimiert. Die Fitnesswerte der besten Individuen sind für die 150 Generationen angegeben.*

Das zuvor genannte Handmodell diente bereits in Kapitel 5 als Beispiel für eine eher schlechte Initialsegmentierung. Darüber hinaus wurde dort anhand eines Krugmodells ein weiteres Beispiel für eine suboptimale Zerlegung gezeigt. Die entsprechende Initialsegmentierung ist im linken Teilbild der Abbildung 7.7 erneut dargestellt. Auch bei diesem Beispiel erweist sich die evolutionäre Patch-Optimierung als sehr hilfreich. Im Teilbild rechts oben der Abbildung 7.7 ist das Ergebnis zu sehen, das sich durch die Optimierung ergibt, wobei allerdings nur die Initial-Seeds und keine Satelliten-Seeds verwendet worden sind. Eine deutliche Verbesserung hat sich am oberen Rand des Kruges ergeben. Gleichzeitig ist allerdings der andere Übergang vom Henkel zum blauen Patch schlechter geworden. Bei Nutzung von Satelliten-Seeds führt die Optimierung hingegen – wie im Teilbild rechts unten dargestellt – zu einem überzeugenden Ergebnis.

Auch anhand anderer Experimente hat sich herausgestellt, dass Satelliten-Seeds einen positiven Einfluss auf die Optimierung der Patches haben. Ohne Satelliten-Seeds tendiert die Optimierung sogar dazu, einzelne Initial-Seeds auf kleinere Teile des Modells zu verschieben, die von einem konkaven Bereich begrenzt werden. Dies führt oft dazu, dass Patches entstehen, die für die erwartete Segmentierungsgranularität zu klein sind. Ein Beispiel ist in Abbildung 7.8 zu sehen. Im linken Teilbild ist ein Initial-Seed auf dem Kopf und ein weiterer auf dem Rumpf des Pferdes angeordnet. Soll nun die Grenze zwischen dem roten und dem grünen Patch durch die Optimierung etwas verbessert werden, führt dies in der Regel zu einer Zerlegung des Modells in ein Ohr und den

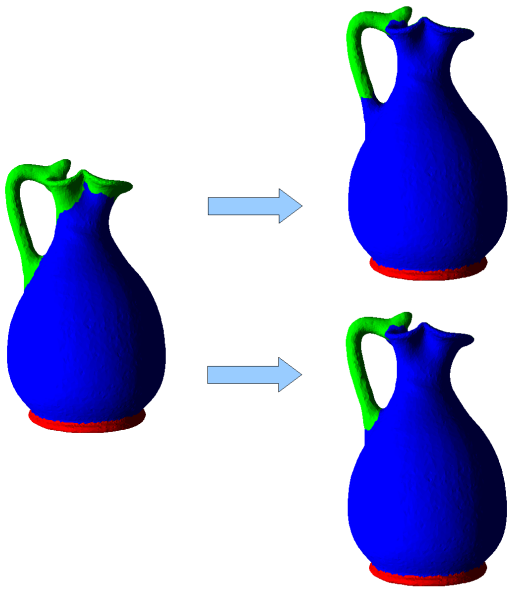


Abbildung 7.7: *Initialsegmentierung (links) sowie die Ergebnisse der Patch-Optimierung ohne (rechts oben) und mit Satelliten-Seeds (rechts unten).*

Rest (mittleres Teilbild). Dies entspricht jedoch nicht mehr annähernd der Segmentierungsgranularität der Initialsegmentierung. Satelliten-Seeds erweisen sich hier als äußerst hilfreich. Im rechten Teilbild der Abbildung 7.8 ist ein typisches Ergebnis nach der Optimierung bei Verwendung von Satelliten-Seeds dargestellt. Hier hat sich die Segmentierungsgranularität durch die Optimierung nicht verändert. Dies liegt daran, dass es bei Verwendung von Satelliten-Seeds sehr unwahrscheinlich ist, dass sämtliche zusammengehörenden Seeds auf eine kleine, von einer konvexen Region umgebene Objektkomponente verschoben werden.

7.6 Fazit

Da nicht sichergestellt werden kann, dass die Initialsegmentierung stets zufriedenstellende Ergebnisse liefert, ist in diesem Kapitel eine Möglichkeit zur evolutionären Optimierung der Patch-Grenzen vorgestellt worden. Der Ansatz basiert auf einer (μ, κ, λ)-ES, bei der auch Parameter mit nicht-kontinuierlichen Wertebereichen vorkommen. Eine Zielfunktion ist definiert worden, die unabhängig vom konkreten Modell eine adäquate Bewertung der Segmentierung ermöglicht. In diese sind zwei Kriterien für gute Segmentierungen eingeflossen, aus denen mit Hilfe der Weighted-Sum-Methode ein einkriterielles Optimierungsproblem konstruiert worden ist.

Die Optimierung kann wahlweise mit oder ohne Berücksichtigung von Satelliten-Seeds erfolgen. Es hat sich gezeigt, dass ohne Satelliten-Seeds bedingt durch die Mutation der Seed-Positionen zu kleine Patches entstehen können. Eine ausschließliche Betrachtung

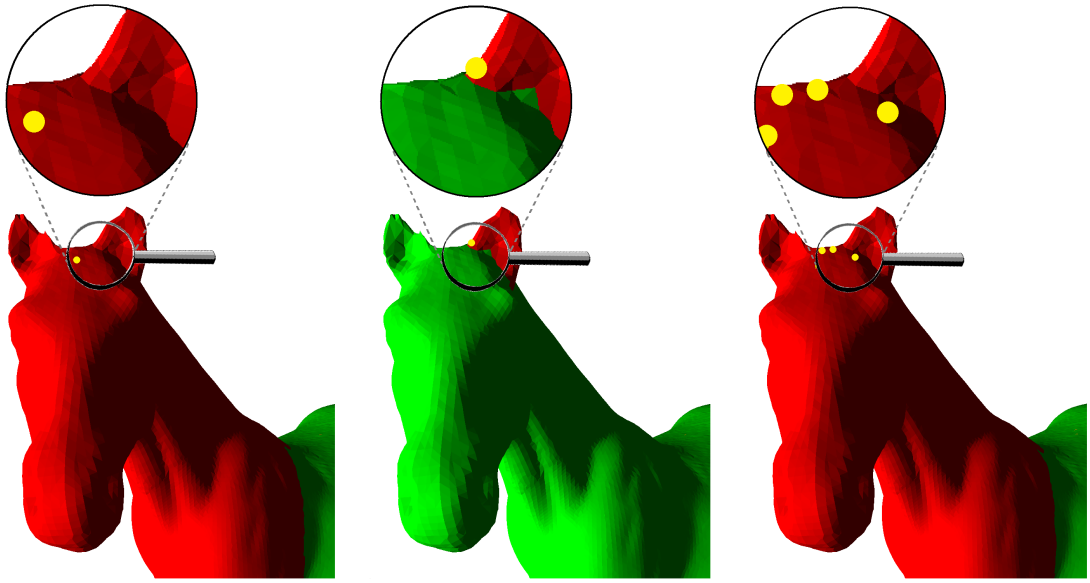


Abbildung 7.8: *Ohne Satelliten-Seeding können durch die Optimierung Situation entstehen, in denen ein Initial-Seed (linkes Teilbild) über eine konkave Kante verschoben wird (Mitte), woraus ein Patch resultiert, das sich stark von dem des Initial-Seeds unterscheidet. Die Verwendung von Satelliten-Seeds reduziert die Wahrscheinlichkeit des Auftretens solcher Situationen (rechtes Teilbild).*

der Initial-Seeds bietet sich jedoch für den ersten Schritt einer zweistufigen Patch-Optimierung an. Sobald die Patches sich nicht mehr gravierend über Grenzen zwischen logischen Komponenten hinweg erstrecken, wird dann in der zweiten Stufe mit Hilfe von Satelliten-Seeds eine feinere Verbesserung der Patch-Ränder in Angriff genommen.

Es wurde gezeigt, dass die evolutionäre Patch-Optimierung ein großes Potential beinhaltet. Mit ihr können aus Patches mit schlecht verlaufenden Rändern bedeutungsvolle Segmente entstehen. Dabei sind die Auswirkungen nicht wie bei einem Boundary-Smoothing-Ansatz zwangsläufig lokal begrenzt; vielmehr kann sich die Gestalt der Patches durch Anwendung der Evolutionsstrategie auch recht stark verändern.

Kapitel 8

Evaluation

Dieses Kapitel befasst sich mit der Evaluation des EvOpSeg-Verfahrens. Dazu werden zunächst in Abschnitt 8.1 Metriken vorgestellt, mit denen sich die Ähnlichkeit zweier Segmentierungen desselben Modells bestimmen lässt. Eine dieser Metriken findet auch bei der Ermittlung geeigneter Mutationsschrittweiten (Abschnitt 8.2) sowie bei der Untersuchung des Einflusses der Satelliten-Seeds (Abschnitt 8.3) Anwendung. Zwei im Jahr 2009 veröffentlichte Benchmarks [BVLD09, CGF09] machen die Ergebnisse von Segmentierungsverfahren quantitativ vergleichbar, abstrahieren dabei aber auch von der visuell wahrgenommenen Segmentierungsqualität. In Abschnitt 8.4 findet eine solche benchmarkbasierte Evaluation für das EvOpSeg-Verfahren sowie ein Vergleich mit den Ergebnissen anderer bekannter Segmentierungsverfahren statt. Dabei wird sowohl das automatische Seeding als auch ein simuliertes manuelles Seeding angewendet, so dass für die Durchführung der Evaluation keinerlei Interaktion notwendig ist. Das simulierte manuelle Seeding wird dann in Abschnitt 8.5 einem echten manuellen Seeding gegenübergestellt. Nach diesen benchmarkbasierten Betrachtungen wird in Abschnitt 8.6 visuell untersucht, für welche Modellarten sich das EvOpSeg-Verfahren offensichtlich besonders gut eignet und bei welchen Modellarten mit Schwierigkeiten bzw. der Notwendigkeit nach überdurchschnittlich vielen manuellen Eingriffen zu rechnen ist. Eine Analyse der für die Patch-Optimierung benötigten Rechenzeit erfolgt schließlich in Abschnitt 8.7.

8.1 Metriken für die Evaluation von Dreiecksnetz-Segmentierungen

Über viele Jahre hinweg sind Segmentierungsverfahren bzw. deren Resultate rein visuell miteinander verglichen worden. So sind mit unterschiedlichen Verfahren für die gleichen oder zumindest ähnliche dreidimensionale Modelle Segmentierungen erzeugt worden, anhand derer die Vor- und Nachteile der Verfahren demonstriert und diskutiert worden sind. Auch wenn die Bewertungen der Ergebnisse in den meisten Fällen nachvollziehbar erscheinen, handelt es sich um keine wirklich objektive Bewertung. Insbesondere

lässt ein solches Vorgehen keine Aussage darüber zu, „um wieviel“ eine Segmentierung besser oder schlechter ist als eine andere. Abhilfe schaffen die Arbeiten von Benhabiles et al. [BVLD09, BVLD10] sowie von Chen et al. [CGF09], in denen Methoden zum Vergleich von Mesh-Segmentierungen vorgestellt werden. Sowohl in [BVLD10] als auch in [CGF09] werden mit dem *Rand-Index* und dem *Konsistenzfehler* zwei konkrete Metriken für die Ähnlichkeit zweier Segmentierungen desselben Modells genannt. Diese kommen auch in mehreren der nachfolgenden Abschnitte zum Einsatz, weswegen sie hier zunächst kurz erläutert werden.

8.1.1 Rand-Index

William M. Rand hat im Jahr 1971 ein Maß zum Vergleich von Daten-Clustern veröffentlicht [Ran71]. Dieser so genannte *Rand-Index* kann auch für die Bewertung von Mesh-Segmentierungen verwendet werden [CGF09], da hierbei die Dreiecksflächen des gegebenen Netzes „geclustert“ werden. Der Rand-Index stellt ein Maß für die Ähnlichkeit der Segmentierungen dar; er misst die Wahrscheinlichkeit, dass zwei beliebige Dreiecksflächen entweder in beiden Segmentierungen zum gleichen Segment oder in beiden zu unterschiedlichen Segmenten gehören.

Zur Ermittlung des Rand-Index werden zwei Segmentierungen S_A und S_B sowie alle möglichen Paare von Dreiecksfacetten betrachtet. Es sei n_A die Anzahl der Paare, deren Dreiecke sowohl in S_A als auch in S_B zu unterschiedlichen Segmenten gehören. Ferner sei n_B die Anzahl der Paare, deren Dreiecke in beiden Segmentierungen jeweils zum selben Segment gehören. Damit gibt es $n_A + n_B$ Dreieckspaare, die in beiden Segmentierungen entweder zum selben Segment oder zu unterschiedlichen Segmenten gehören. Die Gesamtzahl aller möglichen Dreieckspaare beträgt $\binom{n_\Delta}{2}$, wobei n_Δ die Anzahl aller Dreiecksflächen des gegebenen Modells bezeichnet. Damit ist der Rand-Index durch

$$RI_{std}(S_A, S_B) := \binom{n_\Delta}{2}^{-1} (n_A + n_B) \tag{8.1}$$

gegeben, wobei der Wertebereich durch das Intervall $[0, 1]$ gegeben ist. Für weitere Details sei auf [CGF09] verwiesen.

Der Index „std“ in Gleichung (8.1) soll darauf hinweisen, dass es sich um die Standard-Variante des Rand-Index handelt. Bei dieser ist der Wert umso größer, je ähnlicher sich die beiden Segmentierungen sind. Demnach ist ein Segmentierungsverfahren als gut anzusehen, wenn daraus große Rand-Index-Werte resultieren. Dies ist strukturell konträr zur Bestrebung aus Unterabschnitt 8.1.2, einen möglichst kleinen Konsistenzfehler zu erhalten. Aus diesem Grund transformieren Chen et al. in [CGF09] den Rand-Index:

$$RI(S_A, S_B) := 1 - RI_{std}(S_A, S_B). \tag{8.2}$$

Auf diese Weise stehen nun kleine Werte für eine starke Übereinstimmung der beiden Segmentierungen S_A und S_B. Sofern nicht anders erwähnt, ist im Folgenden stets diese Variante gemeint, wenn von „Rand-Index“ die Rede ist.

8.1.2 Konsistenzfehler

Während der Rand-Index als Maß für die Ähnlichkeit zweier Segmentierungen desselben Modells anzusehen ist, steht bei dem auf Martin et al. [MFTM01] zurückgehenden *Konsistenzfehler* oder *Consistency Error* die Differenz der Segmentierungen im Mittelpunkt. Der Konsistenzfehler stellt sicher, dass sich unterschiedliche Segmentierungsgranularitäten nicht negativ auf das Ergebnis auswirken. Die „richtige“ Segment-Anzahl hängt nämlich von der Interpretation des Modells ab. Ein Menschenmodell kann beispielsweise in Rumpf, Kopf, Arme, Beine, Füße und Hände eingeteilt werden, oder feingranularer auch in Rumpf, Kopf, Hals, Oberarme, Unterarme, Oberschenkel, Unterschenkel, Füße, Handflächen und Finger. Keine der beiden Einteilungen ist semantisch „falsch“. Wenn sich die Grenzen der gröberen Segmentierung in der feineren Segmentierung wiederfinden, ist der Wert des Konsistenzfehlers dieser Segmentierungen null.

In [MFTM01] werden zwei Arten von Konsistenzfehlern für die Bild-Segmentierung vorgestellt: der *Globale Konsistenzfehler* oder auch *Global Consistency Error* (*GCE*) sowie der *Lokale Konsistenzfehler* (*Local Consistency Error*, *LCE*). Diese ursprünglich für Bilder vorgesehenen Metriken können auf Dreiecksnetze übertragen werden. Eine entsprechende Beschreibung ist beispielsweise in [CGF09] zu finden, woran sich auch die folgende Zusammenfassung orientiert. Für die Definition des GCE sowie des LCE wird ein so genannter *Local Refinement Error* benötigt, der für zwei Segmentierungen S_A und S_B als

$$LRE(S_A, S_B, t_i) := \frac{|R(S_A, t_i) \backslash R(S_B, t_i)|}{|R(S_A, t_i)|} \tag{8.3}$$

definiert ist. Die beiden Segmentierungen seien dabei als $S_A = \{S_A^1, S_A^2, \ldots, S_A^n\}$ sowie $S_B = \{S_B^1, S_B^2, \ldots, S_B^m\}$ gegeben, wobei n bzw. m die Anzahl der Segmente bezeichnet. Für eine Dreiecksfläche t_i des gegebenen Dreiecksnetzes gibt $R(S, t_i)$ die Menge der Dreiecke desjenigen Segments der Segmentierung S an, das t_i enthält. Damit ist der *Global Consistency Error* definiert als

$$\begin{aligned} GCE(S_A, S_B) := \frac{1}{n_\triangle} \min \Big(& \sum_i LRE(S_A, S_B, t_i), \\ & \sum_i LRE(S_B, S_A, t_i) \Big), \end{aligned} \tag{8.4}$$

wobei $n_\triangle$ die Anzahl der Dreiecksflächen des Netzes ist. Der *Local Consistency Error* ist durch

$$\begin{aligned} LCE(S_A, S_B) := \frac{1}{n_\triangle} \sum_i \min \Big(& LRE(S_A, S_B, t_i), \\ & LRE(S_B, S_A, t_i) \Big) \end{aligned} \tag{8.5}$$

gegeben. Für zwei Segmentierungen S_A und S_B, bei denen alle Segmente von S_A feingranularer sind als die von S_B, hat die unterschiedliche Segmentierungsgranularität

weder einen negativen Einfluss auf den GCE noch auf den LCE. Wenn jedoch einige Teile von S_A feingranularer sind als die korrespondierenden von S_B und gleichzeitig einige andere von S_A gröber sind als die von S_B, dann wird der GCE dadurch negativ beeinflusst, nicht jedoch der LCE. Somit ist der GCE-Wert stets mindestens so groß wie der LCE-Wert.

8.2 Ermittlung geeigneter Mutationsschrittweiten und Parameterwerte

Wie in Abschnitt 7.2 beschrieben nutzt das EvOpSeg-Verfahren einen einzigen Strategieparameter σ, aus dem die Mutationsschrittweiten σ_ζ, σ_η, σ_γ und σ_p abgeleitet werden. Im Folgenden wird ein experimenteller Ansatz beschrieben, der zu einer guten Abstimmung der Mutationsschrittweiten aufeinander führen soll. Zusätzlich wird dabei auch ein problemadäquater Wert für den in der Zielfunktion vorkommenden Faktor Λ ermittelt.

8.2.1 Vorgehen

Durch systematisches Probieren sind erste plausibel erscheinende Initial-Werte[1] für die Mutationsschrittweiten σ_ζ, σ_η, σ_γ und σ_p sowie für den Faktor Λ ermittelt worden. Eine bessere Abstimmung der Werte aufeinander kann mit Hilfe von Segmentierungen erfolgen, die als „optimal“ angesehen werden. Üblicherweise werden solche idealen Segmentierungen, mit denen die EvOpSeg-Ergebnisse bei Verwendung der untersuchten Parameterwerte verglichen werden, manuell erzeugt und sind als *Ground-Truth-Segmentierungen* oder als *Ground-Truth-Daten* (kurz: *GTD*) bekannt. Sowohl in dem Benchmark von Benhabiles et al. [BVLD09] als auch in dem von Chen et al. [CGF09] sind entsprechende Ground-Truth-Daten enthalten. Da der zuletzt genannte Benchmark deutlich umfangreicher als der andere ist, wird er im weiteren Verlauf dieser Arbeit zur Evaluation des EvOpSeg-Verfahrens herangezogen. Somit bietet es sich dann an, zur Bestimmung geeigneter initialer Werte für Λ sowie für die Mutationsschrittweiten Modelle aus dem Benchmark von Benhabiles et al. zu verwenden. Auf diese Weise ist eine Unabhängigkeit von den in der Evaluation verwendeten Modellen sichergestellt.

Eine analytische Bestimmung optimaler Parameterwerte ist offensichtlich nicht möglich. Stattdessen muss experimentell versucht werden, hinreichend gute Werte zu ermitteln. Dafür wird eine Menge M_M von Modellen benötigt, die jeweils auch einen Ground-Truth-Datensatz enthalten. Zur besseren semantischen Abgrenzung von den später folgenden *Evaluationsmodellen* werden die Modelle der Menge M_M hier auch *Analysemodelle* genannt. Für jede zu untersuchende Kombination von Parameterwerten ist für jedes Analysemodell eine entsprechende Segmentierung inklusive einer

[1] Aufgrund der Selbstadaption verändern sich die Schrittweiten im Allgemeinen während eines Optimierungslaufes (vgl. Abschnitt 7.1).

Patch-Optimierung zu erzeugen und das Ergebnis mit dem zugehörigen Ground-Truth-Datensatz zu vergleichen. Da stets eine Patch-Optimierung durchzuführen ist, lassen sich innerhalb einer akzeptablen Zeitspanne nicht beliebig viele Versuche durchführen, wodurch hier beispielsweise eine evolutionäre Optimierung der initialen Werte nicht in Betracht kommt. Stattdessen wird für jeden der genannten Parameter eine Menge von zu untersuchenden Parameterwerten definiert. Diese Mengen werden mit M_ζ, M_η, M_γ, M_p und M_Λ bezeichnet. Für jede mögliche Kombination der Werte dieser Mengen wird dann für jedes Analysemodell der Menge M_M eine optimierte Segmentierung mit dem EvOpSeg-Verfahren erzeugt und die Abweichung von der zugehörigen Ground-Truth-Segmentierung anhand einer adäquaten Metrik – wie dem Konsistenzfehler oder dem Rand-Index – bestimmt. Im weiteren Verlauf wird eine konkrete Belegung von σ_ζ, σ_η, σ_γ, σ_p und Λ mit Werten aus M_ζ, M_η, M_γ, M_p bzw. M_Λ als *Analyseparameterkonfiguration* bezeichnet. Die Erweiterung einer Analyseparameterkonfiguration um ein konkretes Analysemodell wird nachfolgend *Analysekonfiguration* genannt.

In Abbildung 8.1 ist der gesamte Ablauf der Ermittlung geeigneter Parameter schematisch dargestellt. Aktionen, die eine Interaktion verlangen, sind mit einem so genannten User-Symbol versehen, während solche, die ausschließlich vom System durchgeführt werden, ein Zahnrad-Symbol enthalten. Zunächst wird interaktiv ein *Analysedatensatz* bestehend aus den Mengen M_ζ, M_η, M_γ, M_p, M_Λ und M_M definiert. Dabei erfolgt für jedes Modell aus M_M auch eine manuelle Anordnung geeigneter Initial-Seeds. Unter „geeignet“ ist in diesem Zusammenhang zu verstehen, dass pro Ground-Truth-Segment genau ein Initial-Seed vorhanden ist, der sich innerhalb dieses Segments befindet und eher zentral als in der Nähe des Segmentrandes liegt. Nach Erstellung des Analysedatensatzes ist keine weitere Interaktion mehr erforderlich: Automatisch wird die erste Analysekonfiguration samt zugehörigem, mit Initial-Seeds versehenem Analysemodell geladen. Die Initial-Seeds werden dann um Satelliten-Seeds ergänzt. Anschließend findet eine evolutionäre Patch-Optimierung gefolgt von einer Bewertung der optimierten Patches statt. Die genannten automatischen Schritte werden für jede im Analysedatensatz enthaltene Konfiguration durchgeführt und die Bewertungsergebnisse entsprechend gespeichert. Aus den Ergebnissen lässt sich eine optimale Kombination der im Analysedatensatz enthaltenen Parameterwerte für die Mutationsschrittweiten und Λ ableiten. Diese Werte werden im Folgenden *ideale Analyseparameterkonfiguration* genannt.

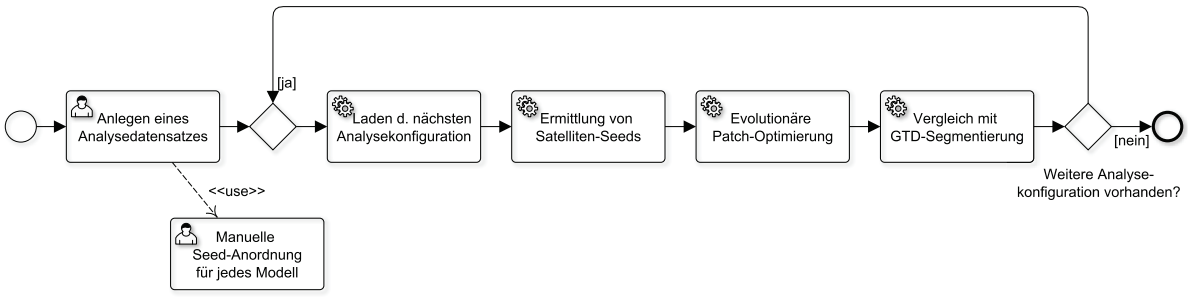


Abbildung 8.1: *Ablauf der Bestimmung geeigneter Parameterwerte.*

8.2.2 Auswertung

Entsprechend dem zuvor beschriebenen Vorgehen ist anhand geeigneter Analysemodelle eine Auswertung zur Ermittlung geeigneter Mutationsschrittweiten sowie eines geeigneten Wertes für den Zielfunktionsparameter Λ erfolgt. Darauf wird hier näher eingegangen.

Wahl der Analysemodelle

Die Ermittlung geeigneter Parameter soll anhand mehrerer Analysemodelle erfolgen. Diese Modelle haben hinreichend komplex zu sein, so dass sie eine Analyse des Zusammenspiels der Parameter ermöglichen. Deswegen sind einige Modelle wie beispielsweise die von Tischen, die für das EvOpSeg-Verfahren aufgrund klarer konkaver Konturen im Bereich des Übergangs zwischen den einzelnen Komponenten auch für die initial gewählten Parameterwerte offensichtlich kein großes Problem darstellen, nicht berücksichtigt worden. Abbildung 8.2 zeigt fünf Modelle aus dem in [BVLD09] angesprochenen Benchmark, die für die Auswertung geeignet zu sein scheinen und die somit herangezogen worden sind. Auch die verwendeten Ground-Truth-Segmente sind der Abbildung zu entnehmen. Das Dinosauriermodell hatte eine deutlich größere räumliche Ausdehnung als die anderen Modelle. Um alle potentiell möglichen Seiteneffekte zu vermeiden, die aus diesem Größenunterschied resultieren könnten, ist dieses Modell für die Auswertung zunächst entsprechend verkleinert worden.

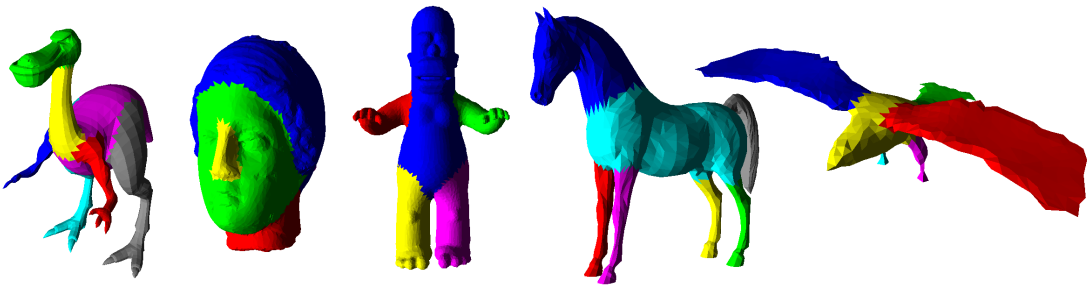


Abbildung 8.2: *Verwendete Analysemodelle und Ground-Truth-Segmente.*

Ausgangswerte für die Parameter und Mutationsschrittweiten

Für jedes Individuum ist eine Patch-Berechnung notwendig. Einen maßgeblichen Einfluss auf die Gestalt der Patches haben dabei die Werte der Parameter von ζ und η. Deswegen werden die zugehörigen Mutationsschrittweiten σ_ζ und σ_η in diesem Abschnitt variiert, so dass darüber entsprechend auch auf ζ und η Einfluss genommen wird. Bei der Wahl der Werte von ζ und η ist die räumliche Ausdehnung des Modells zu berücksichtigen. Dies liegt daran, dass die Modellausdehnung einen direkten Einfluss

auf die geodätische Distanz hat, die den ersten Summanden der verwendeten Distanzfunktion (Gleichung (7.4)) bildet, während dies bei den beiden anderen Summanden nicht der Fall ist. Empirisch sind

$$\zeta := 5000 \cdot Modellausdehnung \quad \text{sowie} \tag{8.6}$$

$$\eta := 15 \cdot Modellausdehnung \tag{8.7}$$

als gute Startwerte ermittelt worden. In der Zielfunktion (Gleichung (7.9)) gibt der Parameter Λ das Verhältnis zwischen den beiden Teilzielen „Konkave Segmentgrenzen“ und „Kurze Segmentgrenzen“ an. Die Wahl hat einen nicht unerheblichen Einfluss auf das Ergebnis der Optimierung. Bei einigen im Vorfeld untersuchten Modellen hat sich für die oben genannten Werte von ζ und η

$$\Lambda := 0.3 \tag{8.8}$$

als sinnvoller Wert für die Konvexkombination herausgestellt. Für andere Parameterkombinationen braucht dieser Wert hingegen nicht zwangsläufig geeignet zu sein.

Der Strategieparameter σ aus Gleichung (7.1) wird mit 1.0 initialisiert. Sein Wert wird sich im Laufe des Optimierungsprozesses durch die Selbstanpassung verändern. In dem hier betrachteten Fall der einfachen Mutation leiten sich die Mutationsschrittweiten σ_ζ, σ_η, σ_γ und σ_p von σ ab. Bei einer ersten empirischen Untersuchung haben sich folgende Mutationsschrittweiten als sinnvoll erwiesen:

$$\sigma_\zeta := 0.05 \cdot \zeta \cdot \sigma \tag{8.9}$$

$$\sigma_\eta := 0.05 \cdot \eta \cdot \sigma \tag{8.10}$$

$$\sigma_\gamma := 0.3 \cdot \sigma \tag{8.11}$$

$$\sigma_p := 3 \cdot \sigma. \tag{8.12}$$

Hierbei wird σ_p auf die nächste ganze Zahl gerundet und gibt die Mutationsschrittweite in „Anzahl an Dreiecken“ an. In die Mutationsschrittweiten σ_ζ und σ_η fließen auch die Parameter ζ bzw. η ein. Hintergrund ist, dass die Werte dieser Parameter vergleichsweise groß sein können und bei größeren Werten auch die Mutationsschrittweite bzw. die Standardabweichung dementsprechend größer sein sollte als bei kleineren. Die in den Gleichungen (8.6) bis (8.12) genannten Werte waren, sofern nicht anders angegeben, auch Grundlage für alle bis hierhin vorgestellten Ergebnisse.

Untersuchte Werte für die Mutationsschrittweiten und für Λ

Eine experimentelle Bestimmung geeigneter Parameterwerte nimmt üblicherweise recht viel Zeit in Anspruch (vgl. auch [Hei07, S. 58]). Deswegen werden in dieser Arbeit nur wenige Parameterwerte untersucht, wodurch sich allerdings bereits 2160 zu untersuchende Analysekonfigurationen ergeben.

Die Parameter ζ und η definieren bei der Patch-Berechnung die Verhältnisse der unterschiedlichen Distanzterme zueinander. Da die initialen Werte der beiden Parameter

relativ groß sind, erscheint es plausibel, bei der Ermittlung geeigneter Mutationsschrittweiten den Mengen M_ζ und M_η mehr Elemente zuzuweisen als den übrigen. Im vorliegenden Fall sind für die Schrittweiten σ_ζ und σ_η jeweils vier Werte untersucht worden, für die beiden anderen Mutationsschrittweiten σ_γ und σ_p jeweils drei. Im Einzelnen sehen die entsprechenden Mengen wie folgt aus:

$$M_\zeta := \left\{ x_\zeta \cdot \zeta \cdot \sigma \mid x_\zeta \in \{0.005, 0.05, 0.25, 0.5\} \right\} \tag{8.13}$$

$$M_\eta := \left\{ x_\eta \cdot \eta \cdot \sigma \mid x_\eta \in \{0.005, 0.05, 0.25, 0.5\} \right\} \tag{8.14}$$

$$M_\gamma := \left\{ x_\gamma \cdot \sigma \mid x_\gamma \in \{0.3, 0.9, 1.5\} \right\} \tag{8.15}$$

$$M_p := \left\{ x_p \cdot \sigma \mid x_p \in \{3, 6, 9\} \right\}. \tag{8.16}$$

Die Werte aus M_p sind größer gewählt als die aus M_γ, da sich bereits eine Veränderung des Seed-Gewichtes, die betragsmäßig kleiner als 0.5 ist, signifikant auf die Gestalt der Patches auswirken kann, während eine Mutationsschrittweite $\sigma_p < 1$ aufgrund der diskreten Verschiebung von Seeds bei der Positionsmutation sicherlich nicht zum Erfolg führt. Insbesondere bei Modellen, die aus sehr vielen Polygonen bestehen, kann ein größerer Wert von σ_p durchaus förderlich sein, da in solchen Fällen teilweise eine Seed-Verschiebung um lediglich wenige Dreiecke kaum Auswirkungen auf die Patches hat.

Wie oben erwähnt scheint $\Lambda = 0.3$ in vielen Fällen eine gute Wahl zu sein. Dennoch werden hier auch für diesen Parameter der Zielfunktion verschiedene Werte untersucht. Die Menge M_Λ sieht wie folgt aus:

$$M_\Lambda := \{0.1, 0.3, 0.6\}. \tag{8.17}$$

Ergebnisse

Ausgangslage für die hier aufgeführten Ergebnisse sind die oben genannten fünf Analysemodelle sowie die Mengen der zu untersuchenden Parameterwerte aus den Formeln (8.13) bis (8.17). Die evolutionäre Patch-Optimierung wurde mit einer $(3, 5, 15)$-ES durchgeführt, wobei jeweils 100 Generationen ausgewertet worden sind. Jeder zuvor manuell angeordnete Initial-Seed wurde dabei mit fünf Satelliten-Seeds versehen. Zur Bewertung der Segmentierungsergebnisse wird die Abweichung der berechneten Segmentierung von der zugehörigen Ground-Truth-Segmentierung mit Hilfe des Rand-Index bestimmt. Insgesamt liegt also für jede der 2160 Analysekonfigurationen (bestehend aus 432 Parameterkonstellationen für jedes der fünf Analysemodelle) ein Rand-Index-Wert vor.

Die Rand-Index-Werte unterschiedlicher Modelle können nicht direkt miteinander verglichen werden, da bei komplexen Modellen üblicherweise selbst für gute Segmentierungen der RI-Wert höher ist als bei einfacheren Modellen. Zudem ist bei komplexen Modellen der Unterschied zwischen den RI-Werten der besten und der schlechtesten Parameterkonstellation häufig deutlich größer als bei einfacheren Modellen. Somit würde bei der Bildung des arithmetischen Mittels der RI-Werte über die fünf Modelle hinweg

(für eine gegebene Parameterkonstellation) das komplexere Modell den durchschnittlichen RI-Wert überproportional stark beeinflussen.

Anstatt den Rand-Index direkt zu betrachten, wird für jede Kombination aus einem Analysemodell und einer Parameterkonstellation ermittelt, um die „wievielt-beste“ Segmentierung dieses Modells es sich in dieser Evaluation handelt. Dadurch entsteht ein Ranking der Parameterkonstellationen für das betrachtete Modell. Im weiteren Verlauf wird in diesem Zusammenhang von einem *Rang* der Parameterkonstellation gesprochen, der sich allerdings nur auf die durchgeführte Untersuchung bezieht und bei einem neuen Evaluationslauf durchaus abweichen kann. Die Parameterkonstellation mit dem niedrigsten RI-Wert für das betrachtete Modell erhält den Rang 1. Zu beachten ist, dass die Ränge nicht eindeutig zu sein brauchen. Gleiche RI-Werte für unterschiedliche Parameterkonstellationen sind durchaus möglich und bei der hier durchgeführten Auswertung auch gelegentlich aufgetreten. Alle Parameterkonstellationen mit gleichen RI-Werten erhalten dann denselben Rang.

Nachdem die Ränge der Parameterkonstellationen für jedes einzelne Modell bekannt sind, wird ein *durchschnittlicher Rang* gebildet, indem für eine konkrete Konstellation von Parameterwerten die Ränge der fünf Analysemodelle arithmetisch gemittelt werden. Die 20 besten Ergebnisse bezüglich des durchschnittlichen Ranges sind der nach diesem Kriterium sortierten Tabelle 8.1 zu entnehmen. Darin sind für die Analysemodelle die ersten zehn Ränge grün und die folgenden zehn gelb hervorgehoben. Alle Ränge ab 200 sind rot hinterlegt. Beim Adlermodell haben zahlreiche Parameterkonstellationen zu demselben Segmentierungsergebnis geführt, so dass sie den gleichen Rang erhalten haben. Dies deutet darauf hin, dass mit jedem von ihnen dasselbe lokale Optimum gefunden worden ist.

Anhand von Tabelle 8.1 wird deutlich, dass offensichtlich keine einzige der untersuchten Kombinationen von Parameterwerten stets „gute“ Ergebnisse liefert. Insgesamt treten bei den bezüglich des durchschnittlichen Ranges besten 20 Konstellationen 15 rot markierte schlechte Ränge auf. Die 18. Konstellation führt für zwei Modelle zu dem besten Segmentierungsergebnis. Dafür ergeben sich aber für zwei andere Modelle mit den Rängen 284 und 202 vergleichsweise schlechte Resultate. Auch die erste Konstellation führt zu zwei sehr guten Rängen. Beim Homer-Modell liegt der Rang gerade noch im ersten Drittel.

Die beiden besten durchschnittlichen Ränge liegen noch relativ nahe zusammen. Ein größerer Sprung von mehr als 20 Rängen entsteht allerdings zwischen dem zweiten und dem dritten Tabelleneintrag. Somit sollte vor allem den ersten beiden eine besondere Beachtung zukommen. Für den zweiten Eintrag spricht, dass der schlechteste (cyan in einer eigenen Spalte hervorgehoben) der fünf einzelnen Rang-Werte deutlich besser ist als beim ersten Eintrag. Darüber hinaus handelt es sich bei ihm sogar um den „besten schlechtesten Rang“ unter allen 432 untersuchten Parameterbelegungen. Dies ist für die Wahl einer geeigneten Parameterbelegung durchaus interessant, da mit dem EvOpSeg-Ansatz angestrebt wird, bei möglichst vielen unterschiedlichen Modellarten zufriedenstellende Ergebnisse zu erhalten.

x_ζ	x_η	x_γ	x_p	Λ	Rang (Büste)	Rang (Pferd)	Rang (Adler)	Rang (Dino)	Rang (Homer)	bester Rang	schlechtester Rang	durchschnittl. Rang
0.25	0.05	1.5	3	0.6	3	6	39	61	134	3	134	48.6
0.5	0.005	0.3	6	0.6	95	32	39	36	68	32	95	54
0.25	0.5	0.3	3	0.6	31	17	39	275	9	9	275	74.2
0.5	0.25	0.9	3	0.3	28	3	39	36	288	3	288	78.8
0.5	0.5	0.3	3	0.1	17	213	39	61	73	17	213	80.6
0.05	0.05	0.9	3	0.6	27	5	39	146	189	5	189	81.2
0.25	0.25	0.3	3	0.1	85	64	39	239	17	17	239	88.8
0.005	0.5	0.9	6	0.3	65	124	39	164	82	39	164	94.8
0.25	0.5	1.5	6	0.6	26	18	39	36	357	18	357	95.2
0.005	0.005	0.3	3	0.3	194	38	39	61	150	38	194	96.4
0.005	0.05	1.5	3	0.6	153	20	39	253	29	20	253	98.8
0.05	0.005	0.9	6	0.6	32	45	39	392	4	4	392	102.4
0.005	0.5	0.3	3	0.6	395	8	39	36	39	8	395	103.4
0.25	0.25	0.3	6	0.6	53	203	39	36	193	36	203	104.8
0.5	0.05	1.5	3	0.1	113	60	39	61	257	39	257	106
0.05	0.25	1.5	6	0.6	191	48	39	248	7	7	248	106.6
0.05	0.25	0.9	3	0.6	91	36	39	314	54	36	314	106.8
0.25	0.005	1.5	9	0.6	1	1	284	202	48	1	284	107.2
0.05	0.25	0.3	3	0.1	10	248	39	186	53	10	248	107.2
0.05	0.5	0.3	6	0.1	23	377	3	59	78	3	377	107.8

Tabelle 8.1: *Die ersten 20 Ergebnisse mit dem besten durchschnittlichen Rang. Grün markiert sind alle Ränge bis maximal 10, gelb die Ränge 11 bis 20 und rot alle Ränge ab 200. Cyan hinterlegt ist der „beste schlechteste Rang“ (bezogen auf die fünf Modelle) aller 432 Parameterbelegungen.*

Ein Nachteil bei der Betrachtung des Ranges ist, dass dieser keine direkten Rückschlüsse auf die Qualität der Segmentierung erlaubt. Die zu einem hohen Rang gehörende Segmentierung kann mitunter nur unwesentlich schlechter sein als die der Parameterkonstellation mit Rang 1. Andererseits können die Rand-Index-Werte von Segmentierungen, die zu direkt aufeinanderfolgenden Rängen geführt haben, auch stark voneinander abweichen, was bedeutet, dass die Qualität der einen Segmentierung als deutlich besser anzusehen ist als die der anderen. Aus diesem Grund wird hier eine *relative Güte* eingeführt, welche die Segmentierungsqualität besser berücksichtigt als der Rang. Mit „relativ“ soll dabei angedeutet werden, dass die Güte nur bezogen auf die aktuelle Untersuchung des Parameterdatensatzes, d.h. für die aktuell erhaltenen Rand-Index-Werte der 432 untersuchten Parameterkonstellationen gilt. Für ein konkretes Modell erhält die Parameterbelegung, die zu dem kleinsten Rand-Index-Wert r_{min} geführt hat, die relative Güte 1, unabhängig davon, wie weit das Ergebnis von einer wirklich optimalen Segmentierung entfernt ist. Entsprechend erhält die Parameterbelegung, die zu dem größten Rand-Index-Wert r_{max} für das Modell geführt hat, die relative Güte 0. Für jede andere Belegung ergibt sich die relative Güte aus dem zugehörigen Rand-Index-Wert r durch die lineare Interpolation $(r_{max} - r)/(r_{max} - r_{min})$.

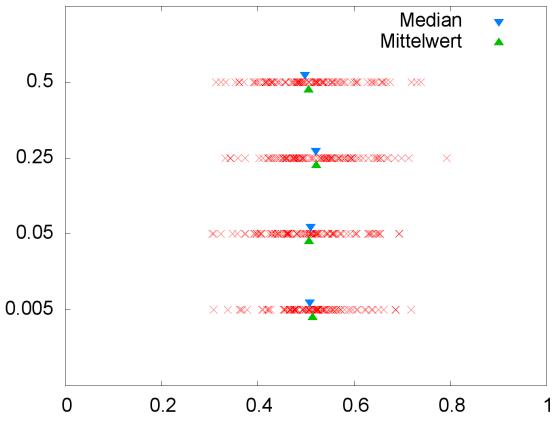
Ähnlich wie beim Rang lässt sich auch die *durchschnittliche relative Güte* als arithmetisches Mittel der relativen Güten für die fünf Modelle bestimmen. Tabelle 8.2 gibt die 20 Parameterkonstellationen mit den besten durchschnittlichen relativen Güten an, wobei die Werte aus Darstellungsgründen auf drei Nachkommastellen gerundet sind. Ähnlich wie zuvor sind auch hier gute Ergebnisse grün (relative Güte größer als 0.9) bzw. gelb (relative Güte zwischen 0.8 und 0.9) hervorgehoben. Außerdem gibt es auch hier viele Parameterkonstellationen, die für mindestens ein Analysemodell zu einer Segmentierung mit einer relativen Güte, deren Wert hier kleiner als 0.5 ist, geführt haben (rot markiert).

Der erste Eintrag aus Tabelle 8.1 liegt auch hier vorne. Obwohl die durchschnittlichen Güten vom zweiten und dritten Eintrag der Tabelle nicht allzu stark voneinander abweichen, ist die Streuung der fünf Güte-Werte der einzelnen Modelle bei dem zuletzt genannten deutlich geringer. Es handelt sich dabei um den Eintrag, der bereits in Tabelle 8.1 absolut gesehen den „besten schlechtesten Rang“ hatte. Der cyan hervorgehobene Wert 0.675 gibt die schlechteste relative Güte an, die bei der Untersuchung der fünf Modelle für diese Parameterbelegung ermittelt worden ist. Für keine andere Parameterbelegung hat sich während der Auswertung ein höherer Wert für die schlechteste relative Güte ergeben, so dass 0.675 die global gesehen „beste schlechteste relative Güte“ darstellt, die während der Untersuchung erzielt worden ist.

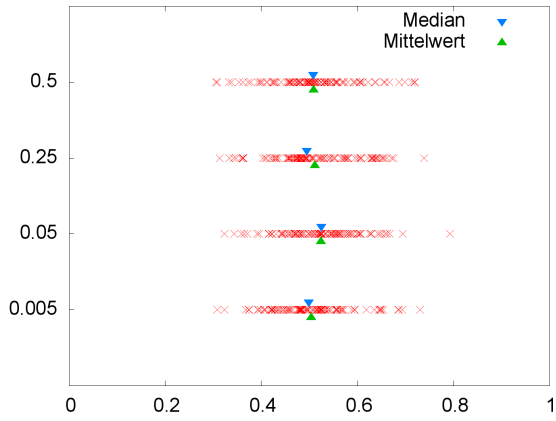
Es kann nicht davon ausgegangen werden, dass der Einfluss der einzelnen Parameter unabhängig voneinander ist. Deswegen ist eine Betrachtung der Auswirkungen eines einzelnen Parameters nur bedingt aussagekräftig. Trotzdem ist in Abbildung 8.3 für jeden der fünf Parameter separat veranschaulicht, wie sich eine Variation dieses Parameters auf die Verteilung der relativen Güte auswirkt, um zu schauen, ob sich auch darin Auffälligkeiten aus den Tabellen 8.1 und 8.2 widerspiegeln. Dabei ist in jedem Teilbild für jedes der 432 erzielten Ergebnisse eine Markierung aufgetragen, der zu entnehmen ist, welcher Parameterwert zu welcher durchschnittlichen relativen Güte

x_ζ	x_η	x_γ	x_p	Λ	rel. Güte (Büste)	rel. Güte (Pferd)	rel. Güte (Adler)	rel. Güte (Dino)	rel. Güte (Homer)	beste rel. Güte	schlechteste rel. Güte	durchschnittl. rel. Güte
0.25	0.05	1.5	3	0.6	0.991	0.882	0.753	0.746	0.59	0.991	0.59	0.792
0.5	0.25	0.9	3	0.3	0.88	0.897	0.753	0.748	0.416	0.897	0.416	0.739
0.5	0.005	0.3	6	0.6	0.675	0.799	0.753	0.748	0.677	0.799	0.675	0.73
0.5	0.5	0.3	3	0.1	0.925	0.504	0.753	0.746	0.67	0.925	0.504	0.719
0.005	0.5	0.3	6	0.1	0.897	0.374	0.918	0.746	0.657	0.918	0.374	0.719
0.25	0.5	1.5	6	0.6	0.884	0.829	0.753	0.748	0.354	0.884	0.354	0.714
0.25	0.5	0.3	3	0.6	0.876	0.831	0.753	0.123	0.911	0.911	0.123	0.699
0.05	0.05	0.9	3	0.6	0.883	0.884	0.753	0.387	0.563	0.884	0.387	0.694
0.05	0.005	0.9	6	0.6	0.875	0.749	0.753	0.099	0.991	0.991	0.099	0.694
0.005	0.5	0.3	3	0.6	0.289	0.872	0.753	0.748	0.773	0.872	0.289	0.687
0.005	0.005	0.3	3	0.3	0.57	0.779	0.753	0.746	0.583	0.779	0.57	0.686
0.25	0.005	1.5	9	0.6	1	1	0.411	0.283	0.732	1	0.283	0.685
0.5	0.25	0.3	3	0.1	0.657	0.406	0.753	0.746	0.813	0.813	0.406	0.675
0.25	0.25	0.3	6	0.6	0.772	0.524	0.753	0.748	0.558	0.772	0.524	0.671
0.25	0.05	0.3	6	0.3	0.573	0.65	0.796	0.901	0.417	0.901	0.417	0.668
0.5	0.5	0.3	3	0.3	0.309	0.664	0.753	0.746	0.862	0.862	0.309	0.667
0.005	0.05	0.3	9	0.6	0.708	0.65	0.411	0.966	0.573	0.966	0.411	0.662
0.5	0.25	0.3	6	0.3	0.877	0.462	0.753	0.748	0.463	0.877	0.462	0.660
0.5	0.05	1.5	3	0.1	0.642	0.708	0.753	0.746	0.444	0.753	0.444	0.659
0.5	0.5	0.3	6	0.1	0.55	0.526	0.753	0.877	0.581	0.877	0.526	0.657

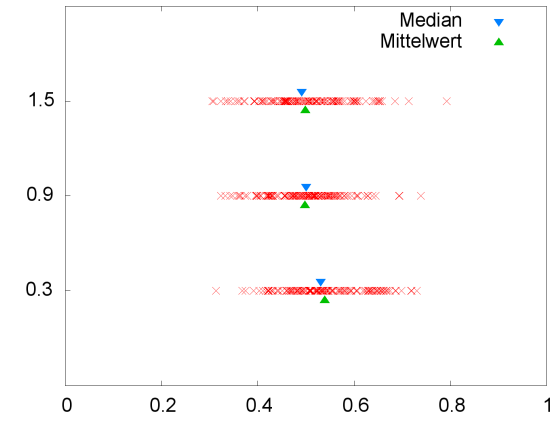
Tabelle 8.2: *Die 20 Ergebnisse mit der besten durchschnittlichen relativen Güte. Grün markiert sind für die fünf Modelle die Werte der relativen Güte, die größer als 0.9 sind, gelb solche zwischen 0.8 und 0.9. Bei einer schlechten relativen Güte (Wert kleiner als 0.5) ist der Wert rot hinterlegt worden. Bei dem cyan markierten Wert handelt es sich um die „beste schlechteste relative Güte“ (bezogen auf die fünf Modelle), die bei den 432 untersuchten Parameterbelegungen erzielt worden ist.*



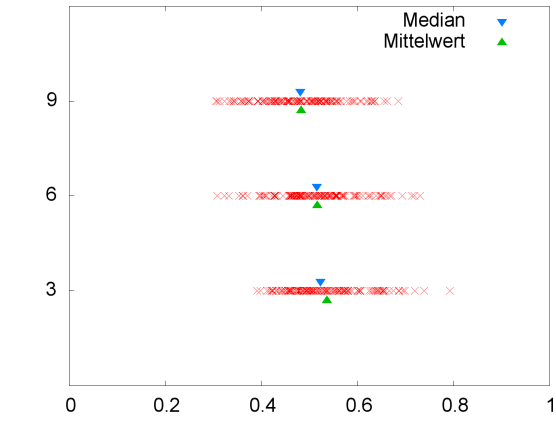
(a) Verteilung der Güte-Werte in Abhängigkeit von x_ζ.



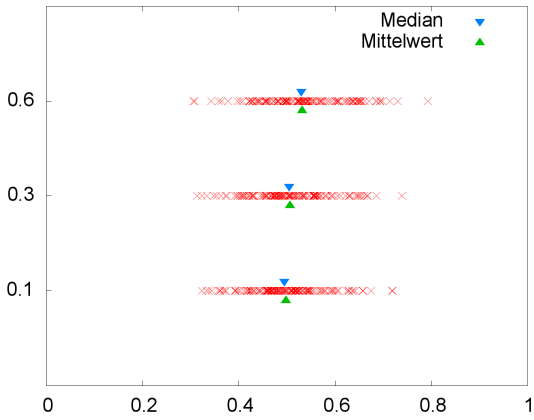
(b) Verteilung der Güte-Werte in Abhängigkeit von x_η.



(c) Verteilung der Güte-Werte in Abhängigkeit von x_γ.



(d) Verteilung der Güte-Werte in Abhängigkeit von x_p.



(e) Verteilung der Güte-Werte in Abhängigkeit von x_Λ.

Abbildung 8.3: *Veranschaulichung der Verteilung der durchschnittlichen relativen Güte-Werte (x-Achse) in Abhängigkeit von den einzelnen Parameterwerten (y-Achse).*

geführt hat. Auf der x-Achse ist die relative Güte aufgetragen, auf der y-Achse sind es die untersuchten Werte des jeweilig betrachteten Parameters.

Die Verteilungen scheinen für x_ζ und x_η keine eindeutigen Rückschlüsse zu erlauben; die Mittelwerte bzw. Mediane der relativen Güten für die untersuchten Parameterwerte sind dort relativ ähnlich. Nebenbei sei an dieser Stelle noch einmal erwähnt, dass es sich um eine relative Güte handelt, die aus dem besten sowie schlechtesten RI-Wert konstruiert worden ist. So sagt der Wert der relativen durchschnittlichen Güte nur bedingt etwas über die Segmentierungsqualität aus, d.h. eine relative Güte von 0.5 kann durchaus zu einem sehr guten Segmentierungsergebnis gehören. Nach Konstruktion der relativen Güte sollte es auch nicht verwundern, dass Mittelwert und Median stets um 0.5 herum liegen.

Nach Abbildung 8.3 ist $x_\gamma = 0.3$ offensichtlich im Durchschnitt besser geeignet, als die anderen beiden Werte. Dieser Wert liegt auch der Parameterbelegung zugrunde, die zum besten schlechtesten Rang sowie zur besten schlechtesten relativen Güte geführt hat. Zudem kommt $x_\gamma = 0.3$ in Tabelle 8.2 mit 13 Mal deutlich überdurchschnittlich oft vor, wohingegen $x_\gamma = 1.5$ dort lediglich viermal vertreten ist. Die Lösung mit dem besten durchschnittlichen Rang nutzt hingegen den hier am schlechtesten erscheinenden Wert $x_\gamma = 1.5$.

Bei der Verteilung der x_p-Werte in Abbildung 8.3 ist deutlich zu erkennen, dass der Bereich, in dem sich die Gütewerte häufen, für $x_p = 3$ weiter rechts liegen als für die anderen beiden Werte. Allerdings sind die Median-Werte für $x_p = 3$ und $x_p = 6$ sehr ähnlich und klar besser als für $x_p = 9$. Auch diese Tatsache spiegelt sich in Tabelle 8.2 wider. Dort kommt $x_p = 3$ zehnmal, $x_p = 9$ hingegen nur zweimal vor.

$x_\Lambda = 0.6$ scheint besser geeignet zu sein, als die beiden niedrigeren Werte. Dieser Wert gehört auch zur Parameterbelegung mit der besten durchschnittlichen Güte sowie zu der mit der besten schlechtesten Güte. Der Wert $x_\Lambda = 0.6$ kommt mit zwölf- bzw. zehnmal auch in den Tabellen 8.1 bzw. 8.2 überproportional häufig vor.

Aufgrund der zuvor aufgezeigten Punkte wird in dieser Arbeit die Parameterkonstellation, die sowohl zum besten schlechtesten Rang als auch zur besten schlechtesten relativen Güte geführt hat, als *ideale Analyseparameterkonfiguration* gewählt.

8.3 Ermittlung einer geeigneten Satelliten-Seed-Anzahl

Neben den im vorangegangenen Abschnitt behandelten Segmentierungsparametern und Mutationsschrittweiten hat auch die Anzahl der Satelliten-Seeds einen Einfluss auf die Qualität der nach einer Patch-Optimierung erhaltenen Segmentierung. Bei zu wenigen Satelliten-Seeds ist zu erwarten, dass das Ergebnis häufig aufgrund fehlender Freiheitsgrade nicht zufriedenstellend ist. Mit steigender Anzahl von Satelliten-Seeds nimmt hingegen die benötigte Laufzeit zu, ohne dass die weiteren Freiheitsgrade zwangsläufig

zu einem besseren Verhalten der evolutionären Patch-Optimierung zu führen brauchen. Im Folgenden wird im Rahmen einer genaueren Untersuchung versucht, eine gute Heuristik für die Satelliten-Seed-Anzahl zu erhalten.

8.3.1 Vorgehen

Zur Untersuchung des Einflusses der Satelliten-Seed-Anzahl werden ähnlich wie auch schon in Abschnitt 8.2 konkrete Experimente durchgeführt, wobei hier allerdings ausschließlich die Anzahl von Satelliten-Seeds variiert wird. Der prinzipielle Ablauf, der starke strukturelle Ähnlichkeiten zu dem aus dem vorangegangenen Abschnitt aufweist, ist in Abbildung 8.4 veranschaulicht. Auch hier kommt ein Analysedatensatz zum Einsatz, der zur sprachlichen Unterscheidung von dem bisherigen aber *Satelliten-Analysedatensatz* genannt wird. Dieser beinhaltet neben den Mengen M_ς, M_η, M_γ, M_p, M_Λ und M_M aus Abschnitt 8.2 auch eine Menge M_s, die alle zu testenden Satelliten-Seed-Anzahlen enthält. Außer M_s bleibt aber nur die Menge M_M der Analysemodelle unverändert. Alle anderen Mengen werden zu jeweils einelementigen Mengen reduziert, indem abgesehen von den jeweiligen Parameterwerten der *idealen Analyseparameterkonfiguration* (vgl. Abschnitt 8.2) alle anderen Werte ersatzlos entfernt werden. Dies bedeutet, dass zu jeder Kombination aus einer in M_s enthaltenen Satelliten-Seed-Anzahl und einem der bereits zuvor verwendeten und mit Seed-Punkten versehenen Analysemodelle eine Berechnung optimierter Patches mit anschließender Bewertung ihrer Qualität durch einen Vergleich mit dem zugehörigen Ground-Truth-Datensatz stattfindet.

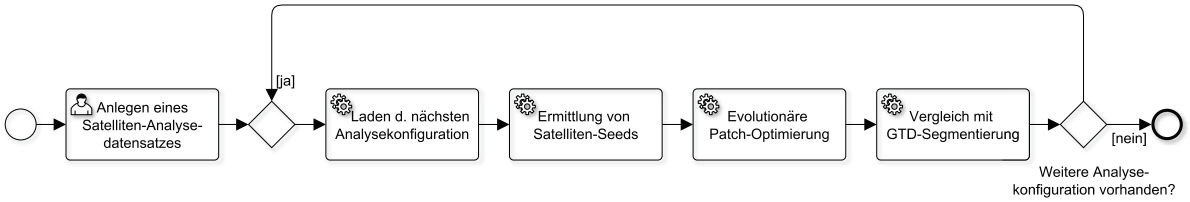


Abbildung 8.4: *Ablauf der Bestimmung einer geeigneten Anzahl von Satelliten-Seeds.*

8.3.2 Auswertung

Bei der Untersuchung zur Ermittlung einer geeigneten Anzahl von Satelliten-Seeds ist, wie auch schon zuvor bei der Ermittlung geeigneter Parameterwerte, eine $(3, 5, 15)$-Evolutionsstrategie für jeweils 100 Generationen zum Einsatz gekommen. Im Rahmen der Auswertung wurden zwischen null und neun Satelliten-Seeds verwendet, so dass die Menge M_s der zur Auswahl stehenden Satelliten-Seed-Anzahlen durch $M_s := \{0, \ldots, 9\}$ gegeben ist. Die Rand-Index-Werte der Ergebnisse sind der Tabelle 8.3 zu entnehmen. Dabei sind je Modell und Satelliten-Seed-Anzahl drei Auswertungen erfolgt und die

Ergebnisse anschließend arithmetisch gemittelt worden. Grün hervorgehoben sind die besten drei Durchschnittswerte je Modell[2].

Anhand dieser Auswertung wird der Nutzen von Satelliten-Seeds deutlich. Kommen ausschließlich Initial-Seeds zum Einsatz, ist der durchschnittliche Rand-Index-Wert stets signifikant größer als bei Verwendung von Satelliten-Seeds. Außerdem ist die Tendenz zu erkennen, dass ab dem dritten Satelliten-Seed eine deutliche Verbesserung zu erwarten ist. Nach Tabelle 8.3 bietet sich insbesondere die Verwendung von drei oder aber von neun Satelliten-Seeds an. Da die benötigte Rechenzeit mit der Satelliten-Seed-Anzahl steigt, werden bei den Auswertungen in den folgenden Abschnitten stets drei Satelliten-Seeds verwendet.

	∅ RI-Wert (Büste)	∅ RI-Wert (Pferd)	∅ RI-Wert (Adler)	∅ RI-Wert (Dino)	∅ RI-Wert (Homer)
ohne Satelliten-Seeds	0.568515	0.375761	0.130351	0.240202	0.526676
1 Satelliten-Seed	0.199495	0.165295	0.039194	0.201652	0.258167
2 Satelliten-Seeds	0.182559	0.148736	0.022039	0.201414	0.288540
3 Satelliten-Seeds	0.113414	0.138400	0.021364	0.161863	0.215417
4 Satelliten-Seeds	0.173847	0.164379	0.022768	0.191734	0.219170
5 Satelliten-Seeds	0.120680	0.147353	0.022768	0.133654	0.186710
6 Satelliten-Seeds	0.175112	0.153086	0.025687	0.129975	0.159923
7 Satelliten-Seeds	0.116834	0.146936	0.031982	0.154102	0.144099
8 Satelliten-Seeds	0.112354	0.16124	0.029804	0.123225	0.162718
9 Satelliten-Seeds	0.095985	0.138625	0.033322	0.099882	0.143383

Tabelle 8.3: *Auswirkung der Satellitenanzahl (drei Versuche pro Modell).*

8.4 Benchmarkbasierte Evaluation bei automatischem Seeding

Für die folgende benchmarkbasierte Evaluation ist der schon zuvor erwähnte *Princeton-Benchmark* genutzt worden [CGF09]. Die Wahl ist auf ihn gefallen, da er mit 380 enthaltenen dreidimensionalen Modellen deutlich umfangreicher als der in [BVLD09] angegebene ist. Zudem enthält er insgesamt 4300 von unterschiedlichen Testpersonen manuell erzeugte Ground-Truth-Segementierungen, also durchschnittlich rund elf Ground-Truth-Datensätze pro Modell.

[2]Bei dem Adlermodell gab es zwei drittbeste Werte.

Üblicherweise sind die Ground-Truth-Segmentierungen eines Modells nicht identisch. Dies ist vor allem in der unterschiedlichen subjektiven Wahrnehmung des Modells und seiner Komponenten begründet. Zwei von unterschiedlichen Personen erzeugte Ground-Truth-Segmentierungen können sich durchaus stark voneinander unterscheiden, obwohl beide Segmentierungen als gut anzusehen sind. Dies kann einerseits an einer unterschiedlichen Granularität liegen, d.h. es ist eine unterschiedliche Anzahl von Segmenten gewählt worden. Andererseits gibt es auch Modelle, bei denen die Semantik der einzelnen Teile nicht klar definierbar ist. Ein Beispiel stellt das Flugzeug aus Abbildung 8.5 dar. Die Tragflächen können, wie im linken Teilbild eingezeichnet, als eine einzige Komponente angesehen werden oder als zwei getrennte. Letzteres ist im rechten Teilbild zu sehen. Dort gehört der mittlere Teil der Tragfläche zum Rumpf des Flugzeugs.

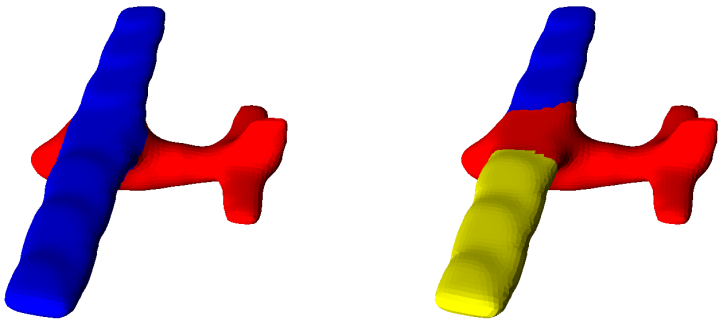


Abbildung 8.5: *Bei einigen Modellen gibt es mehrere semantisch-korrekte Segmentierungen.*

Eine Bewertung der Qualität eines Segmentierungsverfahrens erfolgt mit Hilfe der Ground-Truth-Daten. Für jedes der 380 Modelle wird die durchschnittliche Abweichung der berechneten Segmente von den Segmenten aller zugehöriger Ground-Truth-Daten bestimmt. Dafür existieren unterschiedliche Metriken [BVLD09, CGF09, BVLD10], wie beispielsweise der *Konsistenzfehler* und der *Rand-Index*, die bereits in Abschnitt 8.1 vorgestellt worden sind und hier zur Anwendung kommen. Bei dem in dieser Arbeit beschriebenen automatischen Seeding ist zumindest die Angabe der gewünschten Seed-Anzahl notwendig. Eine solche Anzahl lässt sich jedoch auch aus den Ground-Truth-Daten ableiten, so dass die gesamte Evaluation automatisch erfolgen kann. Damit, sowie mit einer zweiten Form der automatischen Seed-Vergabe, die ein manuelles Seeding simuliert, befasst sich der folgende Unterabschnitt. In Unterabschnitt 8.4.2 erfolgt eine Auswertung der Ergebnisse.

8.4.1 Automatisches Seeding und simuliertes manuelles Seeding

Wie bei anderen Seed-Punkt-basierten Verfahren ist auch beim EvOpSeg-Verfahren die Angabe der gewünschten Patch-Anzahl obligatorisch. Für eine große Anzahl von

Experimenten, wie sie im Rahmen dieser Arbeit durchgeführt worden sind, ist eine automatische Bestimmung der geeigneten Seed-Anzahl für jedes untersuchte Modell äußerst hilfreich. Deswegen sind zwei spezielle automatische Seeding-Varianten, die Informationen aus den gegebenen Ground-Truth-Daten berücksichtigen, bei der Evaluation zum Einsatz gekommen.

Seeding-Variante 1 *Die Anzahl der Initial-Seeds entspricht der (auf eine natürliche Zahl gerundeten) durchschnittlichen Segment-Anzahl aller Ground-Truth-Segmentierungen, die zu dem aktuellen Modell gehören. Die Initial-Seeds werden gemäß der Beschreibung in Kapitel 5 automatisch angeordnet.*

Diese erste Seeding-Variante nutzt also die Ground-Truth-Daten zur Bestimmung der Anzahl von Seed-Punkten. Solche Abschätzungen sind durchaus nicht unüblich; so nehmen beispielsweise Chen et al. die am häufigsten in den zu einem Modell gehörenden Ground-Truth-Daten vorkommende Anzahl an Segmenten als Seed-Anzahl für dieses Modell [CGF09].

Seeding-Variante 1 verteilt die Initial-Seeds annähernd gleichmäßig über das gesamte Modell. Bei der Untersuchung der Qualität des automatischen Seeding in Abschnitt 5.4 wurde gezeigt, dass die Seed-Punkte recht sinnvoll angeordnet werden, wenn es sich um Modelle handelt, deren Komponenten abstehende Teile von ähnlicher Größe sind. Besser geeignet ist ohne Frage eine manuelle Anordnung von Seed-Punkten. Dies ist jedoch bei umfangreichen Benchmarks mit einem nicht unerheblichen manuellen Aufwand verbunden. Deswegen ist in [EM12] das Konzept eines *simulierten manuellen Seeding* eingeführt worden, bei dem ein ideales Seeding-Verhalten nachgebildet wird. Die Idee dahinter ist, die Seeds anhand einer Ground-Truth-Segmentierung S_{GTD} möglichst so anzuordnen, wie es die Person getan hätte, die S_{GTD} manuell erzeugt hat. Wenn die Heuristik zur Platzierung derjenigen ähnelt, die der Entscheidung eines Menschen beim interaktiven Seeding zugrunde liegt, kann dieser Ansatz verwendet werden, um einen Indikator für die durch interaktives Seeding erreichbare Segmentierungsqualität zu bestimmen. Vorteilhaft an diesem Vorgehen ist, dass keine weiteren Testpersonen benötigt werden. Außerdem ermöglicht es auch in gewissem Sinne eine Abschätzung der idealen Segmentierungsqualität bei einer rein automatischen Vergabe der Initial-Seeds. Dies bedeutet natürlich nicht, dass eine Segmentierung entsprechender Qualität bei einem automatischen Seeding auch stets erreicht wird. Dennoch verdeutlicht es die Möglichkeiten und Grenzen, die mit dem EvOpSeg-Ansatz verbunden sind. Die Simulation eines idealen Seeding-Verhaltens führt zu folgender Seeding-Variante:

Seeding-Variante 2 *Für ein gegebenes Benchmark-Modell wird automatisch eine beliebige der zugehörigen Ground-Truth-Segmentierungen gewählt. Für jedes Segment S_i dieser Segmentierung wird ein Initial-Seed bestimmt. Dabei handelt es sich um das Dreieck t_j, das den minimalen euklidischen Abstand vom Schwerpunkt von S_i hat.*

Diese zweite Seeding-Variante, die ein *simuliertes manuelles Seeding* darstellt, basiert auf der Annahme, dass jemand, der eine Ground-Truth-Segmentierung angefertigt hat

beziehungsweise sie als „schlüssig“ ansieht, die Seeds nahe den Schwerpunkten der Ground-Truth-Segmente anordnen würde. Um diese Annahme zu überprüfen, sind für zwanzig Benchmark-Modelle Initial-Seeds manuell platziert und mit den nach Seeding-Variante 2 erfolgten Anordnungen verglichen worden. Visuell wurden dabei im Wesentlichen keine signifikanten Abweichungen festgestellt. Auch die optimierten Segmente waren jeweils recht ähnlich. Eine ausführlichere Analyse erfolgt noch in Abschnitt 8.5.

8.4.2 Auswertung der Segmentierungsqualität

Bei der Auswertung werden drei Varianten des EvOpSeg-Verfahrens untersucht: das *Standard-EvOpSeg-Verfahren*, welches ohne gebundene Mutation und ohne zweistufige Patch-Optimierung abläuft, eine Variante mit gebundener Mutation und eine Variante mit zweistufiger Patch-Optimierung. Zur Beurteilung der Segmentierungsqualität werden dabei der Rand-Index sowie der Konsistenzfehler als Metriken herangezogen.

Betrachtung des gesamten Benchmark

Bei der Analyse des Standard-EvOpSeg-Verfahrens sind für jedes der 380 zugrunde liegenden Benchmark-Modelle zwei Initialsegmentierungen – für jede Seeding-Variante eine – ermittelt und die Patches anschließend mit einer (3,5,15)-ES optimiert worden. Als Terminierungskriterium wurde die Anzahl der Generationen auf 30 begrenzt. Dies erscheint nach einer Untersuchung anhand anderer Modelle ein guter Kompromiss zwischen Segmentierungsqualität und Optimierungszeit zu sein. So ist in Abbildung 8.6 ein typischer Verlauf der Fitnesswerte der besten Individuen zu sehen, der sich für das bereits in Abbildung 4.1 dargestellte, aus 16301 Polygonen bestehende Stanford-Bunny mit sechs automatisch gesetzten Seed-Punkten, fünf Satelliten-Seeds und einer (3,50,15)-ES ergeben hat. Der größte Teil des Fortschritts erfolgt innerhalb der ersten 30 Generationen.

Zum Einsatz kamen bei der Analyse des Standard-EvOpSeg-Verfahrens die Werte der idealen Analyseparameterkonfiguration aus Abschnitt 8.2, die vereinfachend auch *geeignete Parameterwerte* genannt werden, sowie drei Satelliten-Seeds pro Initial-Seed. Abbildung 8.7 zeigt links den Einfluss der evolutionären Patch-Optimierung auf den durchschnittlichen Rand-Index-Wert und stellt rechts die Rand-Index-Werte des EvOpSeg-Verfahrens denen etablierter Segmentierungsverfahren gegenüber[3]. Das EvOpSeg-Verfahren wird hier mit E1 bzw. E2 bezeichnet, wobei die Zahl für die Nummer der verwendeten Seeding-Variante steht. E1 gibt also für die 380 Benchmark-Modelle sowie die 4300 Ground-Truth-Segmentierungen den durchschnittlichen Rand-Index-Wert an, der sich bei Anwendung des EvOpSeg-Verfahrens mit dem automatischen Seeding ergeben hat. Der entsprechende Durchschnittswert für das simulierte manuelle Seeding ist als E2 angegeben. Die etablierten Verfahren basieren auf

[3]Die dargestellten RI-Werte der etablierten Verfahren sind aus [CGF09, KHS10, BAT11] zusammengetragen. Das gleiche gilt auch für die GCE- und LCE-Werte aus Abbildung 8.8.

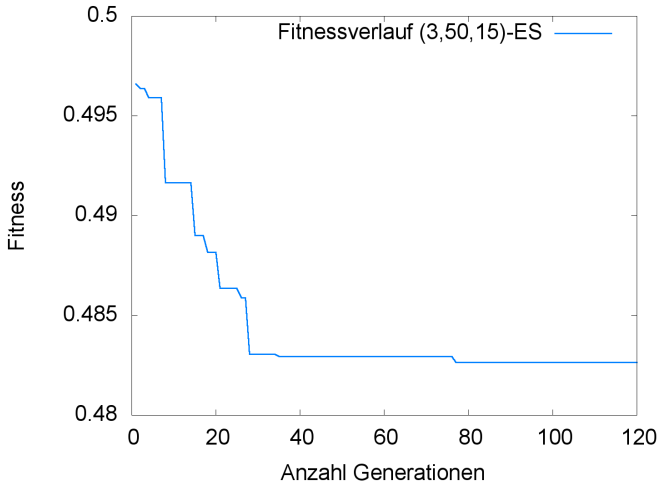


Abbildung 8.6: *Entwicklung der Fitness bei der Patch-Optimierung am Beispiel eines Modells des Stanford-Bunny.*

einem Semi-Supervised-Weighted-Graph-Ansatz (WG) [BAT11], Labeling and Learning (LL) [KHS10], Randomized Cuts (RC) [GF08], der Shape Diameter Function (SD) [SSCO08], Normalized Cuts (NC) [GF08], Core Extraction (CE) [KLT05], Random Walks (RW) [LHMR08], Fitting Primitives (FP) [AFS06] und K-Means Clustering (KM) [STK02]. Dem LL-Ansatz lag dabei für jede einzelne Auswertung eine Trainingsphase zugrunde. Als Trainingsmodelle sind dort jeweils drei andere Modelle derselben Kategorie des Princeton-Benchmark verwendet worden. Die Autoren geben in [KHS10] darüber hinaus auch die Ergebnisse an, die sich bei Verwendung einer größeren Anzahl an Trainingsmodellen ergeben. Werden zuvor für jedes Modell geeignete Werte für die Parameter anhand aller 19 anderen Modelle der Kategorie ermittelt, ergibt sich sogar ein Rand-Index-Wert von rund 0.094. Dieser Wert liegt zwar etwas unterhalb des E2-Wertes aus Abbildung 8.7, allerdings handelt es sich auch um einen vollkommen andersartigen Ansatz, den man sicherlich nicht unreflektiert zum Vergleich heranziehen sollte. Der LL-Ansatz basiert auf einer umfangreichen Trainingsphase mit Modellen, die dem zu segmentierenden ähnlich sind, während das EvOpSeg-Verfahren keine segmentierten Trainingsmodelle benötigt.

Da zu jedem Benchmark-Modell mehrere Ground-Truth-Segmentierungen vorliegen, diese sich jedoch mitunter stark voneinander unterscheiden, liegt selbst bei den bestmöglichen Segmentierungen stets ein positiver RI-Wert vor. Dasselbe gilt auch für die Werte der Konsistenzfehler. Ein Beispiel für eine starke Abweichung zweier möglicher Ground-Truth-Segmentierungen wurde bereits in Abbildung 8.5 angegeben.

Der RI-Wert des EvOpSeg-Verfahrens mit Seeding-Variante 2 ist besser als alle anderen aus Abbildung 8.7. Insbesondere ist im Vergleich zu E1 auch zu erkennen, dass die Wahl guter Seed-Punkte das Segmentierungsergebnis positiv beeinflusst. Dennoch führt auch die erste Seeding-Variante nach der evolutionären Patch-Optimierung zu Ergebnissen, deren durchschnittlicher RI-Wert in etwa dem des Normalized-Cuts-Verfahren

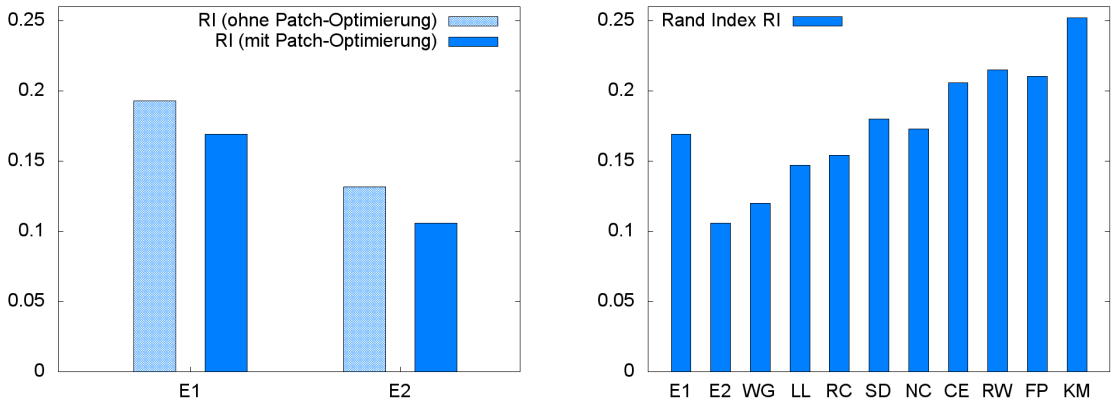


Abbildung 8.7: *Rand-Index-Werte des Standard-EvOpSeg-Verfahrens mit automatischem Seeding (E1) und simuliertem manuellen Seeding (E2).*

entspricht und der – zum Teil sogar deutlich – besser ist als die RI-Werte von fünf weiteren etablierten Verfahren.

Ähnliche Ergebnisse ergeben sich auch bei Betrachtung des Konsistenzfehlers. In Abbildung 8.8 ist links der Einfluss der Patch-Optimierung auf den über die 380 Benchmark-Modelle gemittelten GCE- und LCE-Wert dargestellt. Im rechten Teilbild werden die Konsistenzfehler-Werte der beiden Standard-EvOpSeg-Varianten denen der oben genannten etablierten Segmentierungsverfahren gegenübergestellt. Die GCE-Ergebnisse des EvOpSeg-Verfahrens mit Seeding-Variante 2, des Semi-Supervised-Weighted-Graph-Verfahrens und des Labeling-and-Learning-Verfahrens sind signifikant besser als alle anderen Ergebnisse. Seeding-Variante 1 führt beim EvOpSeg-Verfahren zu Konsistenzfehler-Werten, die zwischen denen des Normalized-Cuts-Verfahrens und des Core-Extraction-Verfahrens liegen. Darüber hinaus liefert die erste Seeding-Variante einen LCE-Wert, der sich kaum von dem des WG-Verfahrens unterscheidet.

Das Labeling-and-Learning-Verfahren und das Semi-Supervised-Weighted-Graph-Verfahren sind bezüglich der beiden Fehlermetriken die besten der etablierten Verfahren. Im Gegensatz zu allen anderen nehmen sie allerdings eine gewisse Sonderrolle ein. Bestandteil des LL-Verfahrens ist eine zeitaufwändige Lernphase, die für Modelle neuer Kategorien vorab durchzuführen ist. Dazu ist das Vorhandensein entsprechender Ground-Truth-Segmentierungen notwendig, die beispielsweise manuell erzeugt worden sind. Auch beim Semi-Supervised-Weighted-Graph-Verfahren sind manuelle Interaktionen zwingend erforderlich, und zwar während des gesamten Segmentierungsprozesses. Die anderen etablierten Verfahren laufen hingegen im Wesentlichen automatisch ab; lediglich die Angabe der gewünschten Segment-Anzahl ist bei manchen Verfahren notwendig. Die genannten Besonderheiten des LL- und des WG-Verfahrens sind bei einem bewertenden Vergleich mit den anderen, automatischen Verfahren zu berücksichtigen.

Neben dem Standard-EvOpSeg-Verfahren sind für das simulierte manuelle Seeding auch das *EvOpSeg-Verfahren mit gebundener Mutation* (mit $\omega = 8$) sowie das *EvOpSeg-Ver-*

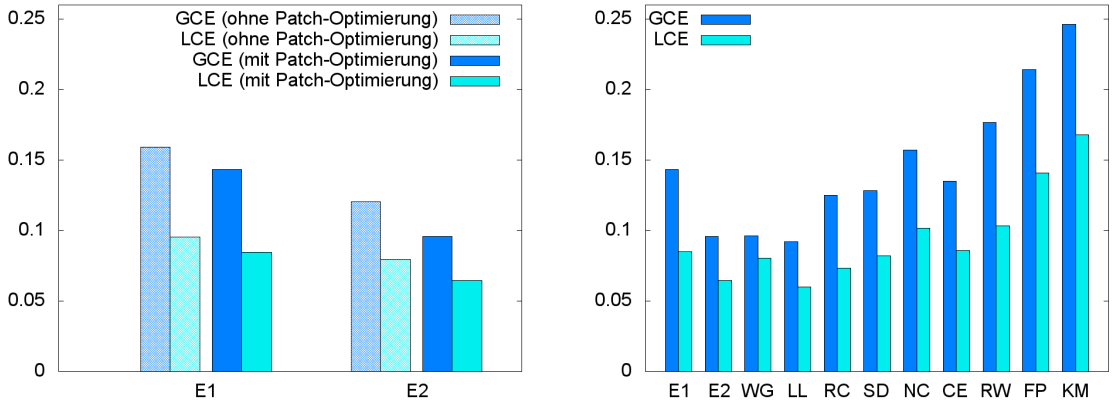


Abbildung 8.8: *Konsistenzfehler-Werte des Standard-EvOpSeg-Verfahrens mit automatischem Seeding (E1) und simuliertem manuellen Seeding (E2).*

fahren mit zweistufiger Patch-Optimierung hinsichtlich des Rand-Index und des Konsistenzfehlers untersucht worden. Die durchschnittlichen Ergebnisse, die sich für die 380 Benchmark-Modelle ergeben, sind in Tabelle 8.4 zusammengefasst. Die Tabelle enthält zusätzlich auch die entsprechenden Ergebnisse, die sich im Rahmen des in [EM12] untersuchten Verfahrens für die beiden Seeding-Varianten – im Folgenden *PPSN-Varianten* genannt – ergeben haben. Den Ergebnissen der PPSN-Varianten liegen die Parameterbelegungen der Formeln (8.6) bis (8.12) zugrunde, die in der vorliegenden Dissertation auch schon als Ausgangswerte für die Ermittlung geeigenter Parameterwerte verwendet worden sind. Außerdem nutzen die PPSN-Varianten fünf Satelliten-Seeds pro Initial-Seed.

		RI	GCE	LCE
autom. Seeding	PPSN-Variante	0.1766	0.1504	0.0915
	Standard-EvOpSeg	0.1691	0.1432	0.0852
	Gebundene Mutation	0.1533	0.1323	0.0790
	Zweistufige Optimierung	0.1448	0.1313	0.0780
sim. man. Seeding	PPSN-Variante	0.1095	0.1016	0.0689
	Standard-EvOpSeg	0.1057	0.0961	0.0649
	Gebundene Mutation	0.1096	0.0929	0.0623
	Zweistufige Optimierung	0.1213	0.0985	0.0653

Tabelle 8.4: *Vergleich der Evaluationsergebnisse.*

Tabelle 8.4 ist zu entnehmen, dass die Ermittlung geeigneter Parameterwerte sowie einer geeigneten Satelliten-Seed-Anzahl zu einer messbaren Verbesserung der Segmentierungsqualität führt. Die RI-, GCE- und LCE-Werte sind bei dem Standard-EvOpSeg-Verfahren sowohl für das automatische Seeding als auch für das simulierte manuelle Seeding besser als bei der entsprechenden PPSN-Variante. So verringert sich beispielsweise bei der zweiten Seeding-Variante der RI-Wert gegenüber der PPSN-Variante um

mindestens 3.4% und der GCE-Wert um mindestens 5.4%. Die Formulierung „mindestens“ soll dabei andeuten, dass sich aufgrund der Unterschiede in den Ground-Truth-Daten stets RI-, GCE- und LCE-Werte ergeben, die echt größer als null sind, was wiederum impliziert, dass die wirkliche prozentuale Verbesserung entsprechend größer ausfällt.

Beim automatischen Seeding führen die gebundene Mutation und die zweistufige Patch-Optimierung nach Tabelle 8.4 zu Verbesserungen gegenüber dem Standard-EvOpSeg-Verfahren. Beim simulierten manuellen Seeding ist dies jedoch nicht der Fall. Dort resultieren aus der zweistufigen Patch-Optimierung Fehlerwerte, die größer sind als die des Standard-EvOpSeg-Verfahrens. Dies legt den Schluss nahe, dass die Konzepte der gebundenen Mutation sowie der zweistufigen Patch-Optimierung in erster Linie dann helfen, wenn Initial-Seeds vorliegen, die nicht optimal angeordnet sind.

Auffällig ist beim simulierten manuellen Seeding der RI-Wert der zweistufigen Patch-Optimierung, der mit 0.1213 deutlich größer ist als der RI-Wert der entsprechenden PPSN-Variante. Gleichzeitig sind jedoch die beiden Konsistenzfehler-Werte etwas kleiner. Anscheinend hängt das damit zusammen, dass die erste Stufe der zweistufigen Optimierung dazu tendiert, Initial-Seeds auf eher kleine Komponenten bzw. auf Regionen des Dreiecksnetzes, die von konkaven Rändern umgeben sind, zu schieben, so dass sich mehrere kleine Patches ergeben. Dies ist vor allem bei solchen Modellen kritisch, die über entsprechende Stellen verfügen, ohne dass diese Stellen jedoch als eigene Komponenten anzusehen sind. Das Hinzufügen von Satelliten-Seeds in der zweiten Stufe bringt dann oft nicht den gewünschten Erfolg, d.h. die unerwünschten kleinen Patches bleiben erhalten. Zu kleine Segmente in einer der zu vergleichenden Segmentierungen führen zu einem schlechten RI-Wert, während dies den Konsistenzfehler nach Konstruktion nur relativ schwach beeinflusst.

Betrachtung der einzelnen Modellkategorien

Bis hierhin ist untersucht worden, wie gut sich das EvOpSeg-Verfahren im Durchschnitt zur Segmentierung eines der Benchmark-Modelle eignet. Von Interesse ist aber auch, für welche Modellkategorien sich das EvOpSeg-Verfahren besonders gut eignet oder bei welchen Modellen es zu Schwierigkeiten kommt. Diese Fragestellung wird im Folgenden behandelt. Dazu werden die durchschnittlichen RI-, GCE- und LCE-Werte für jede der 19 Kategorien ermittelt. Die entsprechenden Werte der etablierten Verfahren haben Chen et al. im Zusammenhang mit der Erstellung ihres Benchmark ermittelt [CGF09]. Die Ergebnisse für das Standard-EvOpSeg-Verfahren und die beiden Varianten mit gebundener Mutation bzw. mit zweistufiger Patch-Optimierung sind in Anhang B aufgeführt. Die präsentierten Ergebnisse des simulierten manuellen Seeding stellen eine Art untere Schranke für die bei einer automatischen Seed-Vergabe zu erwartenden Fehlerwerte dar. Eine echte manuelle Seed-Anordnung führt jedoch zu recht ähnlichen Werten, was im nachfolgenden Abschnitt 8.5 demonstriert wird.

In Abbildung 8.9 sind die RI-Werte, die sich beim simulierten manuellen Seeding ergeben haben, für die 19 Benchmark-Kategorien[4] in Form von Graustufen aufgetragen. Neben den in [CGF09] untersuchten etablierten Segmentierungsverfahren sind auch die mit dem simulierten manuellen Seeding erzielten Ergebnisse für das Standard-EvOpSeg-Verfahren (E2), die gebundene Mutation (EG) und die zweistufige Optimierung (EZ) entsprechend dargestellt. Je besser ein RI-Wert ist, umso heller wird er in der Abbildung dargestellt. Der beste RI-Wert führt zu einem weißen Rechteck in Abbildung 8.9, der schlechteste zu einem schwarzen. Alle zugrunde liegenden Werte für die drei EvOpSeg-Varianten sind in Anhang B.2 angegeben. Es ist deutlich zu erkennen, dass das EvOpSeg-Verfahren bei den Kategorien 5 (Ameisen), 7 (Kraken) und 9 (Teddys) besonders gut abschneidet. Büstenmodelle (Kategorie 16) sind hingegen offensichtlich problematisch; sie scheinen jedoch auch den sieben etablierten Verfahren Schwierigkeiten zu bereiten. Insgesamt schneidet das EvOpSeg-Verfahren im Vergleich zu den anderen Verfahren relativ gut ab; insbesondere ist jedes der in Abbildung 8.9 aufgeführten etablierten Verfahren bezogen auf die RI-Werte in mindestens einer Kategorie deutlich schlechter als das EvOpSeg-Verfahren.

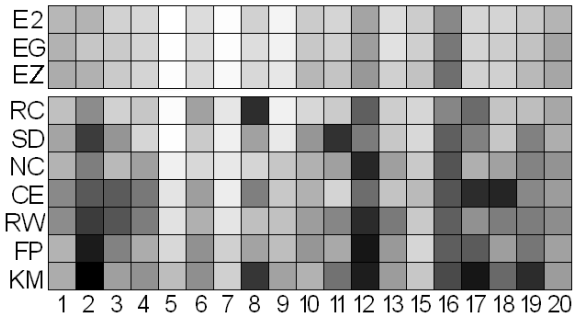


Abbildung 8.9: *Rand-Index-Werte der einzelnen Verfahren für die einzelnen Modellkategorien. Je besser der RI-Wert ist, umso heller ist das entsprechende Quadrat gefüllt.*

Im oberen Teilbild von Abbildung 8.10 sind die GCE-Werte und im unteren Teilbild die LCE-Werte analog zur vorangegangenen Abbildung dargestellt. Auch nach diesen Konsistenzfehlern zu urteilen führt das EvOpSeg-Verfahren offensichtlich für recht viele Kategorien zu guten Ergebnissen. Eine Herausforderung scheinen aber die Kategorien 1 (Menschen), 15 (Armadillo) und 20 (vierbeinige Tiere) darzustellen.

Auffällig ist, dass das Core-Extraction-Verfahren bezüglich des Konsistenzfehlers – anders als bei Betrachtung des Rand-Index – im Mittel deutlich besser abschneidet als die Ansätze, die auf Normalized-Cuts, Random-Walks und Fitting-Primitives basieren (vgl. auch Abbildungen 8.7 und 8.8). Zudem werden für die Armadillomodelle die besten Konsistenzfehler-Werte von dem Core-Extraction-Verfahren geliefert. Dies

[4]Die Nummerierung entspricht der Reihenfolge der Kategorien aus [GBP07], wobei allerdings die Kategorie 14 (spiralförmige Objekte) nicht für eine Segmentierung geeignet ist und somit wie auch in [CGF09] hier keine Berücksichtigung gefunden hat.

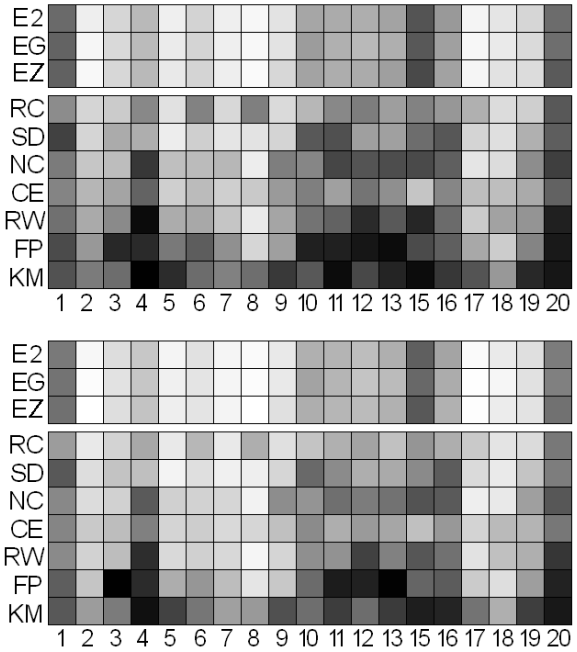


Abbildung 8.10: *GCE-Werte (oberes Teilbild) und LCE-Werte (unteres Teilbild) der einzelnen Verfahren für die einzelnen Modellkategorien. Je besser der GCE- bzw. LCE-Wert ist, umso heller ist das entsprechende Quadrat gefüllt.*

deutet darauf hin, dass dieses Verfahren offensichtlich bei Modellen, die zwar eine feingranulare Oberflächenstruktur aber auch klar erkennbare Extremitäten haben, gute Segmentierungsergebnisse liefert. Allerdings sind die anderen drei genannten Verfahren dem Core-Extraction-Ansatz bezüglich des Rand-Index-Wertes etwas überlegen. Der Grund für diese Diskrepanz zwischen RI- und GCE- bzw. LCE-Wert liegt darin, dass das Core-Extraction-Verfahren für Armadillomodelle vergleichsweise wenige, dafür aber häufig gute Segmente ermittelt. Ein Großteil der Ground-Truth-Daten enhält hingegen recht viele Segmente, so dass sich dies auf den Rand-Index negativ auswirkt, auf den Konsistenzfehler jedoch nicht.

8.5 Benchmarkbasierte Evaluation bei manuellem Seeding

Den benchmarkbasierten Auswertungen aus dem vorherigen Abschnitt lag das automatische Seeding bzw. das simulierte manuelle Seeding zugrunde. Beide kommen vollständig ohne manuelle Interaktionen aus. In diesem Abschnitt sollen nun aber auch Seed-Punkte, die von Testpersonen erzeugt worden sind, bei der Untersuchung des Verfahrens berücksichtigt werden. Prinzipiell sind bei der Anordnung von Seed-Punkten durch Testpersonen unterschiedliche Vorgehensweisen denkbar (vgl. Abbildung 8.11). So ist zu unterscheiden, ob dabei bereits ein Feedback durch Anzeigen

der zugehörigen Initialsegementierung erfolgen soll, oder nicht. In letzterem Fall sollten die Testpersonen zumindest die Instruktion erhalten, jeden Seed-Punkt „irgendwie zentral“ auf der entsprechenden Komponente zu platzieren. Alternativ können auch ganz spezielle Seeding-Regeln vorgegeben werden. Eine Berücksichtigung der evolutionären Patch-Optimierung während des Seeding-Prozesses ist selbstverständlich nur dann angebracht, wenn auch die Patches angezeigt werden.

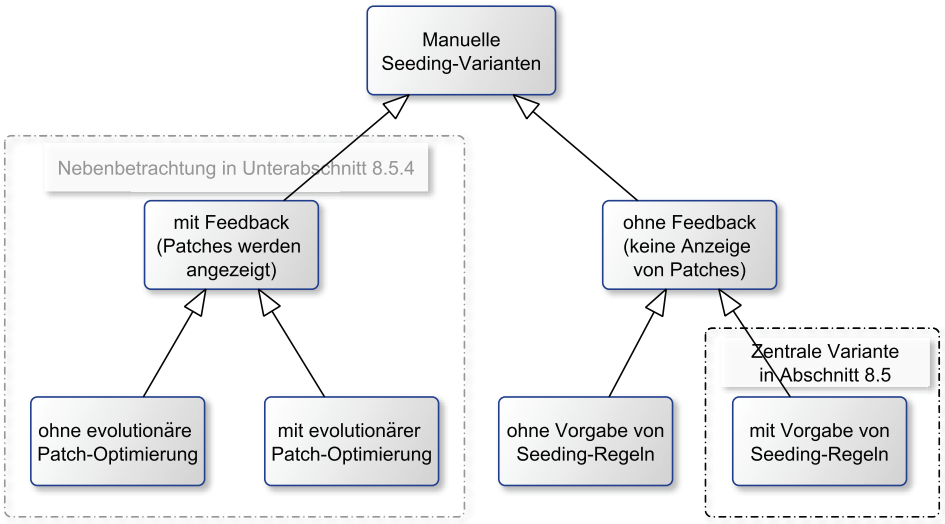


Abbildung 8.11: *Überblick über Varianten des manuellen Seeding.*

Mit den von Testpersonen manuell gesetzten Seed-Punkten werden zwei unterschiedliche Ziele verfolgt: Einerseits wird die Qualität der Segmentierungsergebnisse des EvOpSeg-Verfahrens bei manuell angeordneten Seed-Punkten untersucht. Daneben ermöglichen solche Seeds aber auch eine quantitative Aussage über die Ähnlichkeit zwischen dem simulierten manuellen Seeding und einem echten manuellen Seeding. Beide Ziele lassen sich erreichen, indem die Testpersonen Seeding-Regeln erhalten und die Seeds entsprechend dieser Regeln auf den Modellen anordnen, ohne dabei jedoch die zugehörigen Patches sehen zu können. Diese Vorgehensweise wird in dem vorliegenden Abschnitt verfolgt. Darüber hinaus werden als Nebenbetrachtung bei der Auswertung in Unterabschnitt 8.5.4. für einen kleinen Benchmark-Ausschnitt auch manuell gesetzte Seed-Punkte verwendet, bei deren Platzierung eine Darstellung der Patches stattgefunden hat. Solche „mit Feedback“ gesetzten Seed-Punkte ermöglichen eine Abschätzung der bei interaktiven Eingriffen zu erwartenden Segmentierungsqualität.

8.5.1 Manuelles Seeding durch Testpersonen

Im Folgenden wird das gewählte Vorgehen zur manuellen Platzierung von Seed-Punkten durch Testpersonen beschrieben. Dabei sind insbesondere die beiden folgenden Anforderungen zu erfüllen:

1. Jedes der 380 Modelle des Princeton-Benchmark muss manuell mit Seed-Punkten versehen werden.

2. Es sollen unterschiedliche Testpersonen zum Einsatz kommen.

Die Erfüllung der ersten Anforderung ermöglicht einen kategorieübergreifenden Vergleich mit dem simulierten manuellen Seeding aus Abschnitt 8.4. Andernfalls könnte ein Vergleich höchstens anhand ausgewählter Modellkategorien erfolgen. Eine Verwendung mehrerer Testpersonen soll verhindern, dass eine besonders gut bzw. schlecht agierende Person zu einem verfälschten Ergebnis führt. Offensichtlich ist es nicht sinnvoll, die Modelle den Testpersonen nach Kategorien zuzuweisen. Wenn eine Testperson stets besonders „gute“ oder besonders „schlechte“ Seeds setzt, würde dies dazu führen, dass direkt eine ganze Modellkategorie entsprechend beeinflusst wird. Eine Analyse, für welche Modellkategorien sich das Vorgehen besonders gut eignet, würde so erschwert. Um solche Effekte auszuschließen, erhält jede Testperson genau ein Modell aus jeder Kategorie des Princeton-Benchmark, also insgesamt 19 Modelle[5], die eine *Modellgruppe* bilden. Insgesamt werden dazu 20 Modellgruppen derart konstruiert, dass jedes Benchmark-Modell in genau einer Modellgruppe enthalten ist.

Zur manuellen Anordnung von Seed-Punkten ist im Rahmen dieser Dissertation das Programm `Seeding-Tool` entwickelt worden, das im Anhang näher beschrieben wird. Dieses ermöglicht auf recht einfache Weise das Platzieren von Seeds. Insbesondere können sich die Testpersonen zunächst im Rahmen einer individuellen „Übungsphase“ mit der Bedienung des Programms vertraut machen; erst danach erfolgt die eigentliche Seed-Vergabe. Da primär das Seeding-Verhalten von menschlichen Akteuren und nicht etwa eine „manuelle Patch-Optimierung“ zu untersuchen ist, bietet `Seeding-Tool` keine Möglichkeit zur Anzeige der aus den Seed-Punkten resultierenden Patches an. Somit handelt es sich um ein *Seeding ohne Feedback* (vgl. Abbildung 8.11). Die Seed-Punkte werden direkt in der Modell-Datei gespeichert, was sich durch eine Erweiterung des PLY-Dateiformats[6], das aufgrund solcher Erweiterungsmöglichkeiten für die im Rahmen dieser Dissertation erstellten Programme gewählt worden ist, realisieren lässt.

Die Testpersonen sollen sich nicht um das Laden und Speichern der ihnen zugewiesenen Modelle kümmern müssen. Beim `Seeding-Tool` erfolgt dies, indem die Testpersonen per Button das jeweils lexikographisch[7] folgende oder das vorherige Modell der Modellgruppe aufrufen, also in gewissem Sinne die ihnen zugewiesenen Modelle „durchscrollen“. Bei dem Betätigen eines solchen Button wird das aktuelle Modell automatisch mitsamt der gesetzten Seeds gespeichert und das lexikographisch folgende bzw. vorherige Modell inklusive aller auf ihm platzierten Seeds geladen. So ist es auch möglich, Seed-Anordnungen eines vorher „bearbeiteten“ Modells zu verändern. Das `Seeding-Tool` erfasst für jedes Modell die Zeit, die nach der oben angesprochenen „Übungsphase“ für die manuelle Anordnung der Seeds benötigt wird. Darin fließt die Zeit nicht mit ein, die das Laden und Speichern beansprucht.

[5] Wie bereits erwähnt enthält der Princeton-Benchmark 19 Kategorien mit jeweils 20 Modellen.

[6] PLY steht für *Polygon File Format*. Es ist auch als *Stanford Triangle Format* bekannt und wird von nahezu allen gängigen 3D-Tools unterstützt.

[7] Nach den Dateinamen der Modelle.

Bewusst wurde eine möglichst heterogene Gruppe von Testpersonen gewählt. Ein Großteil von ihnen hatte zuvor keine nennenswerte Affinität zu Problemstellungen aus dem Bereich der dreidimensionalen Computergrafik. Ihnen wurde zunächst anhand eines Beispiels demonstriert, wie aus Seed-Punkten mit dem `EvOpSeg-Tool` Patches entstehen und welchen Einfluss die Anordnung der Seeds auf die Gestalt der Patches hat. Anschließend wurden ihnen folgende drei *Regeln zur Anordnung der Seeds* mit auf den Weg gegeben:

1. Es soll genau ein Seed-Punkt pro „semantischem Teil“ platziert werden, wobei aber keine Vorgaben gemacht worden sind, aus wie vielen Teilen das jeweilige Modell zu bestehen hat.
2. Der Seed-Punkt ist möglichst nah am Schwerpunktes des „Teils“ zu platzieren.
3. Falls es unter 2. mehrere Stellen gibt, die annähernd gleich weit vom Schwerpunkt entfernt sind, ist von diesen eine solche Stelle zu wählen, die vom (gedachten) Rand des „Teils“ maximal weit entfernt ist.

Die Frage, was bei den einzelnen dreidimensionalen Modellen überhaupt „sinnvolle Komponenten“ sind, mussten die Testpersonen für sich selbst beantworten; eine Unterstützung beispielsweise durch Anzeige von zugehörigen Ground-Truth-Segmenten aus dem Princeton-Benchmark war nicht vorhanden. Somit haben die Testpersonen auch eine Entscheidung über die Granularität der erwarteten Segmentierung eigenständig getroffen.

Die Zeit, die von den Testpersonen für die Anordnung der Seeds durchschnittlich pro Modell benötigt wurde, variiert zwischen gut 45 und rund 273 Sekunden, wobei 65% der Personen weniger als 90 Sekunden benötigt haben; die einzelnen Werte sind der Tabelle B.4 im Anhang zu entnehmen. Gemittelt über alle Testpersonen ergibt sich eine durchschnittliche Seeding-Dauer von ca. 100 Sekunden pro Modell.

Einige Testpersonen haben bei komplizierten Modellen gelegentlich den Überblick darüber verloren, an welchen Stellen sie bereits einen Initial-Seed platziert haben, was dazu geführt hat, dass manche Komponenten versehentlich keine Seeds erhalten haben. Eine Anzeige von entsprechenden Segmenten wäre in solchen Fällen sicherlich hilfreich. Dies würde jedoch dem Vorhaben widersprechen, Seeds „ohne Feedback“ setzen zu lassen. Für zukünftige Seedings durch Testpersonen erscheint es allerdings sinnvoll, das Modell auf Knopfdruck transparent erscheinen zu lassen, um so auch hinten liegende Seeds sehen zu können.

Abgesehen von der Tatsache, dass die Seeds nicht durch das Objekt hindurch zu sehen waren, sind die Testpersonen im Wesentlichen ganz gut mit der Bedienung des Programms zurecht gekommen. Auch diejenigen, die zuvor noch nie mit einem 3D-Tool in Berührung gekommen sind, empfanden die Bedienung oft als angenehm und intuitiv. Bei einigen Testpersonen deutete sich mit zunehmender Bearbeitungsdauer allerdings eine gewisse Ermüdungserscheinung an, d.h. die Initial-Seeds wurden bei späteren Modellen nicht mehr so akribisch angeordnet wie bei den zuerst bearbeiteten. Dies zeigte

sich unter anderem auch darin, dass die Modelle nicht mehr ganz so oft gedreht worden sind, sondern häufig alle oder zumindest zahlreiche Initial-Seeds ausgehend von einer einzigen Ansicht des 3D-Modells gesetzt worden sind. Daraus können allerdings auch Seed-Punkte resultieren, die nicht nahe am Schwerpunkt der Komponente liegen, sondern eher mittig auf dem Komponentenausschnitt, den eine Testperson bei der zum Zeitpunkt des Seed-Setzens aktuellen Drehung des Modells sieht.

8.5.2 Bestimmung der Ähnlichkeit von Seed-Konfigurationen

Anhand der von Testpersonen manuell angeordneten Seed-Punkte lässt sich untersuchen, wie ähnlich das simulierte manuelle Seeding einem (in diesem Fall von Testpersonen vorgenommenen) rein manuellen Seeding ist. Sind die Abweichungen gering, spricht dies dafür, dass Seeding-Variante 2 von Seite 144 zur Simulation eines manuellen Seeding gut geeignet ist. Zwei entsprechende Ansätze zum Vergleich beliebiger Seed-Konfiguration werden im Folgenden beschrieben. Dabei ist mit *Seed-Konfiguration* eine Menge von Initial-Seeds gemeint, deren Seed-Gewichte in diesem Zusammenhang allerdings keine Rolle spielen.

Der erste Ansatz zur Bestimmung der Ähnlichkeit zweier Seed-Konfigurationen basiert auf dem geodätischen Abstand korrespondierender Seeds. Darauf wird zunächst eingegangen. Bei symmetrischen Komponenten hat dieser Ansatz jedoch mitunter Schwächen. Abhilfe soll der zweite Ansatz schaffen, der eine Erweiterung des ersten darstellt und Distanzen im $\mathbb{R}^3$ betrachtet.

8.5.2.1 Ansatz 1: geodätischer Hausdorff-Average-Abstand

Ausgangspunkt sind zwei Seed-Konfigurationen, die sich auf dasselbe Modell beziehen. Die Kardinalitäten der beiden Seed-Mengen brauchen jedoch nicht übereinzustimmen. Zur Ermittlung der Ähnlichkeit sind hier im Wesentlichen zwei Teilprobleme zu lösen:

1. Ermittlung der korrespondierenden Seeds der beiden Mengen.
2. Bestimmung eines Wertes für die Abweichung der Seed-Konfigurationen voneinander (anhand der korrespondierenden Seeds).

Das erste Teilproblem kann mitunter als abgewandeltes *Optimal Assignment Problem* angesehen werden. Da die beiden Seed-Mengen allerdings nicht gleichmächtig zu sein brauchen, muss eine optimale Zuordnung derart erfolgen, dass alle Elemente der kleineren Menge einen „Partner“ haben. Beide genannten Teilprobleme lassen sich auch in einem einzigen Schritt mit Hilfe des Hausdorff-Abstandes lösen, der sich in zahlreichen Bereichen der Computergrafik wie beispielsweise zur Bestimmung der Ähnlichkeit von dreidimensionalen Objekten [TLK09] etabliert hat. Dabei wird davon ausgegangen, dass sich die Seeds beider Seed-Mengen gleichzeitig auf demselben Modell befinden.

Für zwei Mengen A und B ist der *einseitige Hausdorff-Abstand* definiert als

$$h(A, B) := \max_{\mathbf{a} \in A}(\min_{\mathbf{b} \in B} d(\mathbf{a}, \mathbf{b})), \tag{8.18}$$

wobei $d(\mathbf{a}, \mathbf{b})$ den euklidischen Abstand der Punkte $\mathbf{a}$ und $\mathbf{b}$ voneinander im $\mathbb{R}^3$ bezeichnet [TLK09]. Da bei der Bestimmung der Ähnlichkeit von Seed-Konfigurationen Abstände zwischen Seed-Punkten, die auf der Oberfläche desselben Modells liegen, zu ermitteln sind und nicht etwa wie in [TLK09] Abstände zwischen unterschiedlichen dreidimensionalen Modellen, wird in dieser Dissertation anstelle der euklidischen Distanz d die approximierte geodätische Distanz d_{geo} aus Abschnitt 2.3 verwendet. Dies verhindert, dass der betrachtete Abstand gering sein kann, obwohl beide Seeds auf weit entfernten Komponenten liegen. Eine solche ungewünschte Situation ist in Abbildung 8.12 exemplarisch dargestellt. Der euklidische Abstand zwischen den beiden auf den Unterschenkeln gelb markierten Punkten ist deutlich geringer als der geodätische Abstand.

Abbildung 8.12: *Beispiel dafür, dass der euklidische Abstand (blau angedeutet) zweier Seed-Punkte deutlich geringer sein kann, als der geodätische Abstand.*

Für zwei Mengen A und B sind die beiden einseitigen Hausdorff-Abstände $h(A, B)$ und $h(B, A)$ im Allgemeinen nicht identisch. Dies gilt insbesondere dann, wenn die Mengen unterschiedlich mächtig sind. Deswegen wird statt des einseitigen ein so genannter *zweiseitiger Hausdorff-Abstand* verwendet (vgl. bspw. [HKKR93]):

$$H(A, B) := \max(h(A, B), h(B, A)). \tag{8.19}$$

Die oben genannte Konstruktion des einseitigen Hausdorff-Abstandes bewirkt, dass bereits ein einziger „Ausreißer“ zu einem verhältnismäßig großen Hausdorff-Abstand führen kann. Für einen sinnvollen Vergleich von Seed-Konfigurationen ist dies allerdings hinderlich. Liegen beispielsweise zwei gleichmächtige Mengen $A = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$

und $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ von Seed-Punkten vor mit $d_{geo}(\mathbf{a}_i, \mathbf{b}_i) = 0$ für $1 \leq i < n$ und $d_{geo}(\mathbf{a}_n, \mathbf{b}_n) = \psi > 0$ und gilt für $1 \leq i < n$ zudem $d_{geo}(\mathbf{a}_i, \mathbf{b}_n) \geq \psi$ sowie $d_{geo}(\mathbf{a}_n, \mathbf{b}_i) \geq \psi$, dann gilt $H(A, B) = \psi$. Den gleichen Wert erhält man aber auch, wenn alle „korrespondierenden“ Seeds den Abstand ψ voneinander haben (und der Abstand zwischen „nicht-korrespondierenden“ Seeds größer als ψ ist). Offensichtlich sind sich die Seed-Mengen jedoch in diesem Fall weniger ähnlich. Deswegen wird hier ähnlich wie in [SELC12] ein Hausdorff-Average-Abstand H_{avg} verwendet, der auf einem einseitigen Hausdorff-Average-Abstand

$$h_{avg}(A, B) := \frac{1}{\|A\|} \cdot \sum_{\mathbf{a} \in A} \min_{\mathbf{b} \in B} d_{geo}(\mathbf{a}, \mathbf{b}) \tag{8.20}$$

basiert. Der (zweiseitige) *Hausdorff-Average-Abstand* ergibt sich dann nach

$$H_{avg}(A, B) := \max(h_{avg}(A, B), h_{avg}(B, A)). \tag{8.21}$$

Der Hausdorff-Average-Abstand soll eine Bestimmung der Ähnlichkeit zweier Seed-Konfigurationen ermöglichen. Da allerdings eine geodätische Distanz verwendet wird, ist dieses Vorgehen nicht invariant gegenüber einer Skalierung des Objekts. Aus diesem Grund wird der Hausdorff-Average-Abstand noch durch

$$H^*_{avg}(A, B) := \frac{1}{d_{ext}} \cdot H_{avg}(A, B) \tag{8.22}$$

normiert, wobei d_{ext} die maximale Ausdehnung des Modells angibt[8]. Sofern nicht anders erwähnt, ist im weiteren Verlauf der Arbeit auch ohne explizite Erwähnung der Normierung – abgesehen vom folgenden Unterabschnitt 8.5.2.2 – stets H^*_{avg} gemeint, wenn vom *Hausdorff-Average-Abstand* gesprochen wird.

8.5.2.2 Ansatz 2: Hausdorff-Average-Abstand bezüglich der Schwerpunkte der Ground-Truth-Segmente

Häufig existieren mehrere (zumindest ungefähr) gleichwertige Positionen, an denen Seeds gemäß der zweiten Seeding-Variante aus Abschnitt 8.4.1 angeordnet werden können. Trotz vergleichbarer Eigenschaften kann der geodätische Abstand zwischen diesen Positionen jedoch sehr groß sein. Exemplarisch wird dies durch Abbildung 8.13 veranschaulicht. Darin soll der elliptische Zylinder ein Ground-Truth-Segment symbolisieren. Offensichtlich gibt es zwei Stellen auf der Mantelfläche, die von dem blauen Schwerpunkt die geringste Entfernung haben (rot markiert). Bei dem zuvor beschriebenen ersten Ansatz wäre der Hausdorff-Abstand zwischen diesen beiden Punkten auf dem Modell recht groß, obwohl sie aber offensichtlich von derselben Qualität sind. Deswegen wird nun ein modifizierter Hausdorff-Average-Abstand eingeführt, der dieses Problem zu vermeiden versucht.

[8] Bei der Realisierung wurden dazu die euklidischen Abstände der Schwerpunkte der Dreiecksflächen voneinander betrachtet.

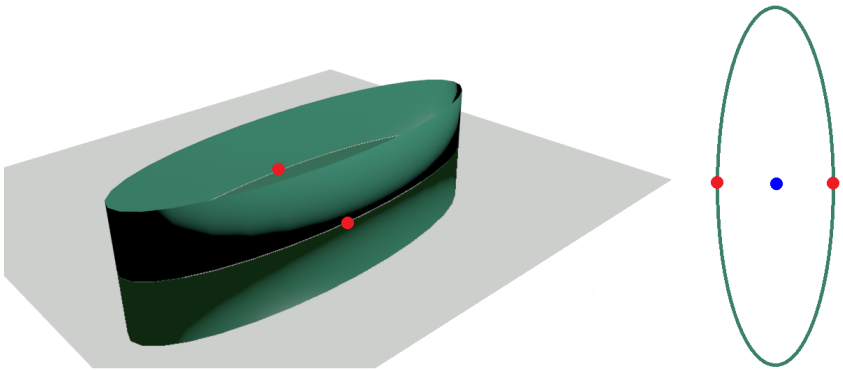


Abbildung 8.13: *Im linken Teilbild wird der offene Zylinder mit elliptischer Grundfläche von einer transparenten Fläche in der Mitte durchdrungen. Der sich ergebende Querschnitt ist im rechten Teilbild dargestellt. Es existieren dort zwei Punkte (rot markiert), die dem blauen Zylinderschwerpunkt am nächsten liegen.*

Dieser zweite Ansatz vergleicht zwei Seed-Konfigurationen anhand der Ähnlichkeit der Seed-Abstände zu den Schwerpunkten von bekannten Segmenten. Als solche Segmente bieten sich die Ground-Truth-Segmente an. Dabei wird diejenige Ground-Truth-Segmentierung herangezogen, die auch beim simulierten manuellen Seeding Verwendung gefunden hat. Des Weiteren werden zur Definition des modifizierten Hausdorff-Average-Abstandes auch die folgenden beiden Hilfsfunktionen benötigt.

$$\text{seg}(\mathbf{x}) := \text{Menge aller Dreiecke, die zu dem GTD-Segment gehören, auf dem } \mathbf{x} \text{ liegt.} \tag{8.23}$$

$$\text{cog}(\mathbf{x}) := \text{Schwerpunkt des GTD-Segments, auf dem } \mathbf{x} \text{ liegt.} \tag{8.24}$$

Diese ermitteln zu einem auf der Oberfläche des Modells liegenden Punkt $\mathbf{x}$ das zugehörige Ground-Truth-Segment bzw. den Schwerpunkt von diesem. Anstelle von Gleichung (8.20) wird nun für die Seed-Punkt-Mengen A und B mit Hilfe der beiden Hilfsfunktionen ein einseitiger Abstand als

$$h_{mod}(A,B) := \frac{1}{\|A\|} \cdot \sum_{\mathbf{a} \in A} \|d(\mathbf{a}, \text{cog}(\mathbf{a})) - d_b(B, \text{seg}(\mathbf{a}), \text{cog}(\mathbf{a}))\| \tag{8.25}$$

definiert. Neben der euklidischen Distanzfunktion d wird dort über d_b auch der minimale Abstand des Schwerpunktes des Segmentes, auf dem der Seed-Punkt $\mathbf{a}$ liegt, zu allen solchen Seed-Punkten der Menge B ermittelt, die ebenfalls auf dem Segment liegen. Die entsprechende Distanzfunktion wird damit für eine Menge von Seed-Punkten M_S, der Menge aller zum aktuellen GTD-Segment gehörenden Netzdreiecke M_Δ sowie einen beliebigen Punkt $\mathbf{y}$ im Raum wie folgt definiert:

$$d_b(M_S, M_\Delta, \mathbf{y}) := \begin{cases} \min\{d(\mathbf{x},\mathbf{y}) \mid \mathbf{x} \in M_S, \Delta(\mathbf{x}) \in M_\Delta\}, & \text{falls } (\Delta(M_S) \cap M_\Delta \neq \emptyset) \\ 0, & \text{ansonsten} \end{cases} \tag{8.26}$$

Dabei ist Δ eine Funktion, die für einen Seed-Punkt das zugehörige Seed-Dreieck und für eine Menge von Seed-Punkten die entsprechende Dreiecksmenge liefert. Unter Verwendung von h_{mod} ist der *zweiseitige modifizierte Hausdorff-Average-Abstand* analog zu Gleichung (8.21) definiert als

$$H_{mod}(A, B) := \max(h_{mod}(A, B), h_{mod}(B, A)). \tag{8.27}$$

Wie bei Ansatz 1 wird auch hier eine Normierung durchgeführt, so dass sich der *normierte zweiseitige modifizierte Hausdorff-Average-Abstand* kanonisch als

$$H^*_{mod}(A, B) := \frac{1}{d_{ext}} \cdot H_{mod}(A, B) \tag{8.28}$$

ergibt, wobei d_{ext} wieder für die maximale Ausdehnung des Modells steht. Im Weiteren Verlauf dieser Arbeit wird der normierte zweiseitige modifizierte Hausdorff-Average-Abstand vereinfachend *modifizierter Hausdorff-Average-Abstand* genannt.

8.5.3 Vergleich mit simuliertem manuellem Seeding

Im Folgenden wird das simulierte manuelle Seeding aus Abschnitt 8.4 mit einem echten manuellen Seeding verglichen. Dies erfolgt durch Bestimmung der Ähnlichkeit von Seed-Konfigurationen, die zu demselben Modell gehören. Um die resultierenden Werte besser interpretieren zu können, wird zusätzlich auch die Ähnlichkeit zwischen den manuell gesetzten Seed-Punkten und denen analysiert, die sich aus dem automatischen Seeding gemäß Seeding-Variante 1 von Seite 144 ergeben.

In Tabelle B.5 im Anhang sind die für die einzelnen Modellgruppen gemittelten Werte der Hausdorff-Average-Abstände sowie der modifizierten Hausdorff-Average-Abstände angegeben. Dabei wird für jede Modellgruppe sowohl das automatische Seeding als auch das simulierte manuelle Seeding mit den von Testpersonen manuell vorgenommenen Seed-Anordnungen verglichen. Die Werte sind für das simulierte manuelle Seeding durchweg signifikant besser als für das automatische Seeding, d.h. die Ähnlichkeit zwischen dem simulierten manuellen Seeding und dem echten manuellen Seeding ist so gesehen deutlich größer als zwischen dem automatischen und dem manuellen. Die vergleichsweise großen Werte für den (nicht-modifizierten) Hausdorff-Average-Abstand sind nicht zuletzt auch auf die oben geschilderte Problematik zurückzuführen, dass es bei einigen Komponenten mehrere in etwa gleichwertige Stellen zur Anordnung von Seeds gibt, die allerdings recht weit voneinander entfernt liegen.

Während Tabelle B.5 Durchschnittswerte für Modellgruppen angibt, veranschaulicht Abbildung 8.14 die gesamte Verteilung aller ermittelten modifizierten Hausdorff-Average-Werte, d.h. für jedes einzelne Modell befinden sich dort zwei Markierungen: eine für den Abstand der manuell gesetzten Seeds zu denen des simulierten manuellen Seeding (obere „Reihe“) und eine für den Abstand zu denen des automatischen Seeding (untere „Reihe“). Der Mittelwert des modifizierten Hausdorff-Average-Abstandes zum automatischen Seeding liegt bei rund 0.1, während es zum simulierten manuellen Seeding rund 0.04 ist. Da es sich um normierte Werte handelt, bedeutet dies,

dass der Unterschied zwischen simuliertem manuellen Seeding und manuellem Seeding bezüglich des modifizierten Hausdorff-Average-Abstandes bei lediglich 4% bezogen auf die Ausdehnung des Modells liegt; der Median ist sogar noch geringer. Dabei ist zu berücksichtigen, dass bei manchen Modellen die Anzahl der von den Testpersonen erkannten Komponenten und damit der platzierten Seeds von denen des simulierten manuellen Seeding abweicht, was sich nach Konstruktion negativ auf den modifizierten Hausdorff-Average-Abstand auswirkt.

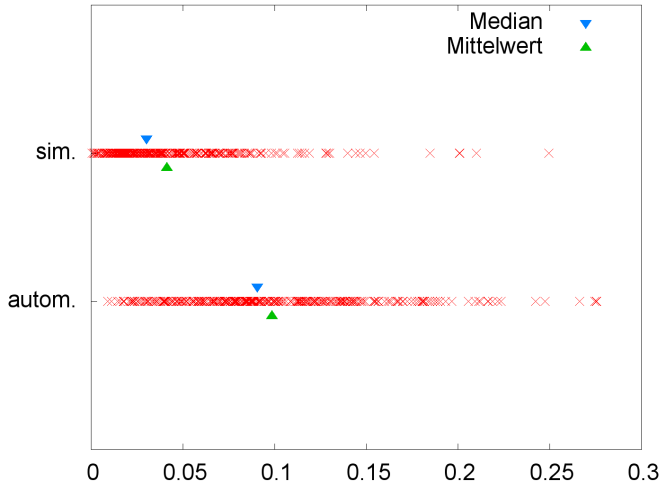


Abbildung 8.14: *Verteilung der Werte des modifizierten-Hausdorff-Average-Abstandes für das automatische Seeding sowie das simulierte manuelle Seeding.*

Eine zufällig ausgewählte Modellgruppe, die Gruppe 10, ist zusätzlich von einer zweiten Testperson bearbeitet worden, um auf diese Weise zu ermitteln, wie ähnlich Seed-Konfigurationen einander sind, die von realen Personen erzeugt werden. Der Hausdorff-Average-Wert beträgt 0.1728, der modifizierte Hausdorff-Average-Wert 0.0393. Damit ist die Ähnlichkeit zwischen dem manuellen Seeding und dem simulierten manuellen im Durchschnitt etwas größer als die zwischen den von den beiden Testpersonen erzeugten Seed-Konfigurationen. Allerdings ist die Abweichung eher gering.

8.5.4 Auswertung der Segmentierungsqualität

Ein Vergleich der manuell gesetzten Seeds mit denen des simulierten manuellen Seeding und des automatischen Seeding ist bereits im vorangegangenen Unterabschnitt erfolgt. An dieser Stelle wird nun auf die Segmentierungsqualität eingegangen, die sich bei Verwendung von manuell gesetzten Seeds ergibt. In erster Linie sind damit die Seeds gemeint, die von den Testpersonen mit dem `Seeding-Tool` auf den Modellen angeordnet worden sind. Darüber hinaus wird als Nebenbetrachtung für zwei Modellgruppen

zusätzlich auch untersucht, welche Segmentierungsqualität bezogen auf den Princeton-Benchmark erreicht werden kann, wenn das `EvOpSeg-Tool` genutzt wird und beliebige Interaktionen erlaubt sind.

Segmentierung auf Basis des Seeding-Tool

Neben den Rand-Index- und Konsistenzfehler-Werten des automatischen Seeding (E1) und des simulierten manuellen Seeding (E2) für das Standard-EvOpSeg-Verfahren sind in den Diagrammen der Abbildung 8.15 auch die entsprechenden Werte angegeben, die sich bei Verwendung der durch Testpersonen manuell angeordneten Initial-Seeds ergeben haben (E3). Die E3 zugrunde liegenden Werte sind der Tabelle B.3 im Anhang zu entnehmen. Es ist zu erkennen, dass der durchschnittliche Rand-Index-Wert beim manuellen Initial-Seeding in etwa dem des simulierten manuellen Seeding entspricht. Die Konsistenzfehler-Werte sind mit rund 0.085 (GCE) bzw. 0.058 (LCE) im Mittel sogar deutlich besser als die des simulierten manuellen Seeding. Tabelle B.3 listet zudem auch die Fehler-Werte bei Verwendung der EvOpSeg-Varianten mit gebundener Mutation bzw. mit zweistufiger Patch-Optimierung auf.

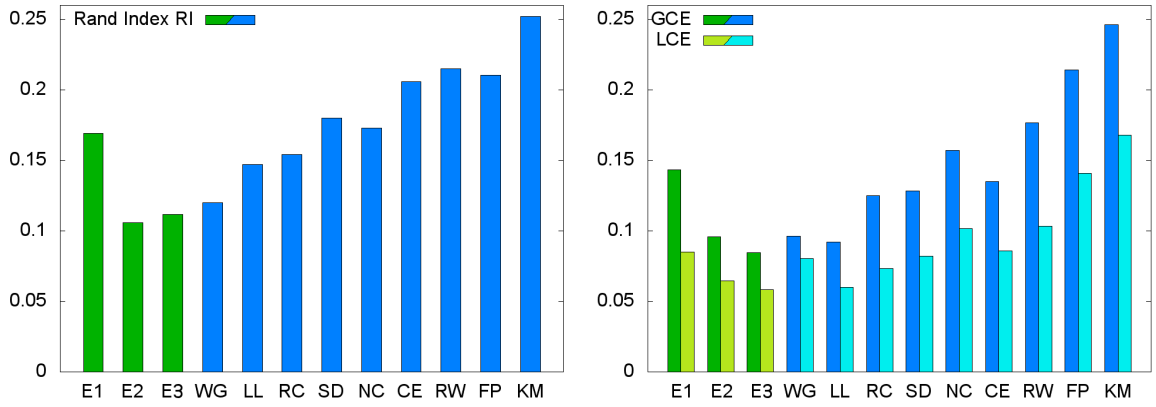


Abbildung 8.15: *Rand-Index- und Konsistenzfehler-Werte des Standard-EvOpSeg-Verfahrens mit automatischem Seeding (E1), simuliertem manuellen Seeding (E2) und echtem manuellen Seeding (E3) im Vergleich zu etablierten Verfahren.*

Für die von den Testpersonen manuell angeordneten Seeds sind die Rand-Index- sowie Konsistenzfehler-Werte, die sich bei Anwendung des Standard-EvOpSeg-Verfahrens ergeben haben, nach Modellgruppen getrennt in Tabelle B.4 angegeben. Ebenfalls in Tabelle B.4 vermerkt ist die Seeding-Dauer in Sekunden, die von der jeweiligen Testperson durchschnittlich pro Modell benötigt worden ist. Die Zeiten variieren teilweise sehr stark. Ein Grund dafür ist, dass manche Testpersonen äußerst akribisch an die Aufgabe herangegangen sind und sehr genau überlegt haben, wo sie Seeds platzieren. Damit einher ging oft auch der Wunsch, feinere und dadurch deutlich mehr Segmente zu erhalten, was eine größere Anzahl an Initial-Seeds und damit eine längere Seeding-Dauer

bedingt. Aber auch die Tatsache, ob sich jemand bereits zuvor mit 3D-Programmen bzw. dreidimensionalen Modellen befasst hatte, spielt sicherlich eine Rolle; beispielsweise taten sich Personen, die umfangreiche Computerspiel-Erfahrung haben, mit der Steuerung etwas leichter als die anderen Testpersonen, und waren somit schneller. Gemittelt über alle Testpersonen wurden pro Modell ca. 100 Sekunden zur Platzierung der Initial-Seeds benötigt.

Interaktive Segmentierung

Im Gegensatz zum `Seeding-Tool` ermöglicht das `EvOpSeg-Tool` eine *interaktive Segmentierung*, bei der auf das dargestellte Ergebnis reagiert werden kann. Beispielsweise ist es möglich, nach einer Patch-Optimierung weitere Seed-Punkte hinzuzufügen oder Seed-Gewichte zu ändern. Auch ein Platzieren von Seed-Linien, die bisher bei der Evaluation keine Rolle gespielt haben, kann interaktiv erfolgen und durchaus hilfreich sein. Für die Modellgruppen 1 und 4 sind interaktive Segmentierungen erfolgt; Tabelle 8.5 gibt die zugehörigen Rand-Index- und Konsistenzfehler-Werte an. Die beiden genannten Modellgruppen wurden gewählt, da die Ergebnisse der ersten Gruppe beim manuellen Seeding sowohl bezüglich des Rand-Index als auch des Konsistenzfehlers vergleichsweise gut sind und die der vierten Gruppe als durchschnittlich bzw. eher schlecht anzusehen sind (vgl. Tabelle B.4). In beiden Fällen wirkt sich die Interaktion positiv aus, wobei die Verbesserung bei der vierten Modellgruppe signifikant größer ist.

Gruppe	RI	GCE	LCE
1	0.0824	0.0717	0.0510
4	0.0704	0.0654	0.0419

Tabelle 8.5: *Auswertung der Segmentierungsqualität bei einer interaktiven Segmentierung.*

8.6 Visuelle Evaluation

Eine Betrachtung der Tabelle B.2 im Anhang legt die Vermutung nahe, dass das EvOpSeg-Verfahren besonders gut geeignet ist für die Kategorien 5 (Ameisen), 6 (Stühle), 7 (Kraken), 8 (Tische), 9 (Teddys), 17 (Mechanik) und 18 (Bolzen). Der relativ große Rand-Index-Wert bei den Tischen resultiert aus der Tatsache, dass bei einigen Ground-Truth-Daten die Tischplatte aus mehreren Teilen besteht und bei anderen aus einem einzigen. Dadurch entsteht ein Fehler, der sich in dem Rand-Index widerspiegelt, nicht jedoch im Konsistenzfehler. Weniger gut scheint das EvOpSeg-Verfahren unter Verwendung des simulierten manuellen Seeding nach Tabelle B.2 bei den Kategorien 1 (Menschen), 12 (Fische), 15 (Armadillo), 16 (Büsten) und 20 (Vierbeiner) zu sein. Eine Betrachtung der zugehörigen Segmentierungsergebnisse bestätigt diese Beobachtungen weitgehend. Allerdings wirken mehrere Segmentierungen der vierbeinigen Tiere (Kategorie 20) deutlich besser, als es die Fehlerwerte aus Tabelle B.2

vermuten lassen. Bei Verwendung der manuell gesetzten Seeds fällt eine visuelle Beurteilung der Segmentierungen im Wesentlichen ganz ähnlich aus. Es wurde aber auch sichtbar, dass das EvOpSeg-Verfahren bei den Krugmodellen (Kategorie 19) überzeugt, wenn manuell gesetzte Seeds verwendet werden, während dort beim simulierten manuellen Seeding neben vielen guten auch ein paar schlechtere Ergebnisse entstehen, da dort einzelne der nach der zweiten Seeding-Regel gesetzten Initial-Seeds direkt in der Nähe eines Komponentenrandes liegen.

Exemplarisch sind in Abbildung 8.16 Segmentierungsergebnisse dargestellt, die sich mit dem Standard-EvOpSeg-Verfahren für die Modelle der ersten Gruppe ergeben, auf denen Initial-Seeds von einer Testperson manuell angeordnet worden sind.

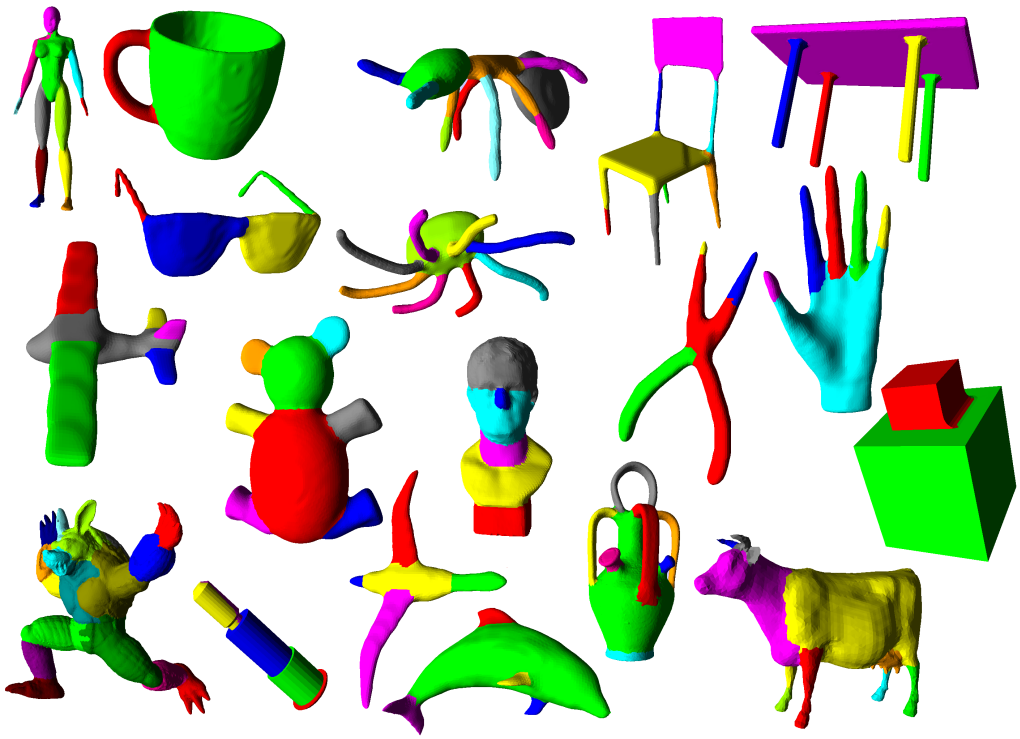


Abbildung 8.16: *Segmentierungsergebnisse für die erste Modellgruppe (manuell gesetzte Initial-Seeds).*

Eine spezielle Betrachtung derjenigen Segmentgrenzen, die als nicht-optimal anzusehen sind, hat gezeigt, dass deren Verläufe dennoch aufgrund der Strukturen der jeweiligen Objekte oft nachvollziehbar erscheinen. Ist beispielsweise das „Bein-Segment“ eines Ameisenmodells zu klein, liegt es in den meisten Fällen daran, dass das Bein aus stark voneinander abgesetzten Gliedern besteht, es also ringsherum stark „eingekerbt“ ist, so dass der Segmentrand mit der Einkerbung zusammenfällt. Somit handelt es sich nicht um ein suboptimales Segment, sondern gewissermaßen um ein optimales, das jedoch nicht der erwarteten Segmentierungsgranularität entspricht. Sobald jedes Bein-Glied einen eigenen Initial-Seed erhält, ist eine erwünschte, feingranularere Segmentierung zu

erwarten. Verallgemeinert kann man also festhalten, dass eine starke Oberflächenstruktur innerhalb einer logischen Komponente zu einem auf den ersten Blick unerwarteten Verlauf der Segmentgrenzen führen kann.

Für die Modelle aus Abbildung 8.16 ist, wie bereits in Unterabschnitt 8.5.4 beschrieben, zusätzlich auch eine interaktive Segmentierung erfolgt. Während die Armadillo- und die Büstenmodelle besondere Herausforderungen für automatische Segmentierungsverfahren darzustellen scheinen, helfen hier wenige interaktive Eingriffe deutlich weiter. In Abbildung 8.17 sind die beiden entsprechenden Modelle aus Abbildung 8.16 jeweils links erneut etwas größer als zuvor dargestellt. Rechts daneben sind Ergebnisse zu sehen, die mit einer interaktiven Segmentierung zu erzielen sind. Insbesondere wurden dabei auch Seed-Linien genutzt, was beim Armadillomodell beispielsweise im Bereich des Übergangs von Kopf und Körper zum Rückenpanzer enorm geholfen hat.

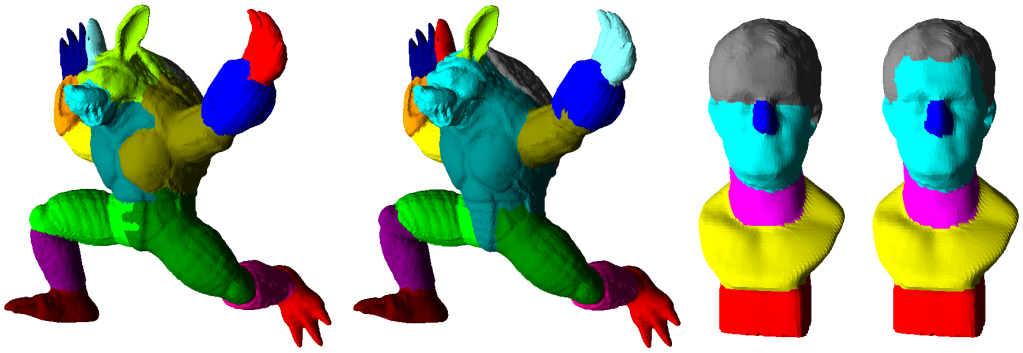


Abbildung 8.17: *Demonstration positiver Auswirkungen einer interaktiven Segmentierung. Jeweils links befindet sich ein Beispiel aus Abbildung 8.16, jeweils rechts das Ergebnis einer interaktiven Segmentierung.*

Anhand der Tischmodelle wird erkennbar, dass beim automatischen Seeding das Strecken oder Stauchen eines Modells einen enormen Einfluss auf die Segmentierungsqualität nehmen kann. Sind die Tischbeine nicht viel kürzer als der Durchmesser der Tischplatte, werden die Seeds auch bei Seeding-Variante 1 recht zuverlässig auf den Beinen angeordnet, so dass sich bedeutungsvolle Segmente ergeben. Dies ist im linken Teilbild der Abbildung 8.18 zu erkennen. Anders sieht es hingegen aus, wenn die Tischbeine recht kurz sind. In einem solchen Fall liegen viele der automatisch gesetzten Seeds über die Tischplatte verteilt, so dass die Patches nicht mehr als bedeutungsvolle Segmente anzusehen sind (rechtes Teilbild).

Gegenüber dem Standard-EvOpSeg-Verfahren hat sich die Variante mit der gebundenen Mutation gelegentlich als vorteilhaft erweisen. Solche Situationen liegen vor, wenn sich Initial-Seeds in der Nähe von Rändern logischer Komponenten befinden. Durch die Bindung zusammengehöriger Seeds aneinander – also die gebundene Mutation – wird in vielen dieser Situationen verhindert, dass einige Seeds nach der Optimierung diesseits und andere jenseits einer solchen Komponentengrenze liegen. Insbesondere bei

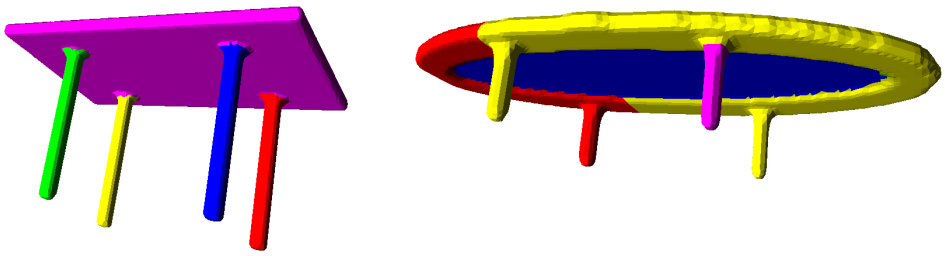


Abbildung 8.18: *Das automatische Seeding funktioniert bei „normalen“ Tischen zuverlässig (links). Sind jedoch die Tischbeine im Verhältnis zur Tischplatte sehr klein, kann es versagen (rechts). In beiden Fällen sind fünf Initial-Seeds automatisch angeordnet worden.*

recht schmalen Komponenten wie beispielsweise dem Hals eines Menschenmodells war die positive Auswirkung der gebundenen Mutation zu erkennen.

In Abbildung 8.19 sind einige Beispiele für die Auswirkung der evolutionären Patch-Optimierung bei Anwendung des Standard-EvOpSeg-Verfahrens zu sehen. Den Beispielen lagen die um Satelliten-Seeds erweiterten Initial-Seeds zugrunde, die Testpersonen manuell auf den Modellen angeordnet haben (vgl. Abschnitt 8.5). In der oberen Bildhälfte sind die Segmente zu sehen, die sich vor Anwendung der evolutionären Patch-Optimierung ergeben. Die entsprechenden Optimierungsresultate in der unteren Hälfte demonstrieren das Potential des EvOpSeg-Ansatzes. Außer bei der rechts unten angeordneten Büste, bei der die Patch-Optimierung offensichtlich keinen positiven Einfluss hat, sind bei allen anderen abgebildeten Beispielen durch die Patch-Optimierung mitunter erhebliche Verbesserungen erzielt worden. Bei dem Beistelltisch links oben läuft das Patch des blauen Beines ohne Patch-Optimierung in die Unterseite der Platte hinein. Nach der Optimierung sind alle Beine perfekt segmentiert. Auch die Grenzen zwischen gelben und blauen Brillenglas, zwischen den Handflächen der beiden Hände und den Fingern sowie zwischen den Flügeln des Vogels und dessen Rumpf werden sichtbar besser. Beim linken Pferd waren zwei Beine vor der Optimierung schlecht segmentiert, beim rechten Pferd das grüne Hinterbein (das rote Patch läuft zu stark in dieses hinein). Außerdem wirkt sich die Optimierung beim rechten Pferd positiv auf den Übergang von Kopf zum Hals aus. Im Gegensatz zum Stuhl und dem bolzenartigen Modell, bei denen die Verbesserungen recht gut zu erkennen sind, gibt es bei dem Armadillomodell eher kleine Fortschritte; so läuft beispielsweise der gelbe Oberarm vor der Patch-Optimierung in den Rückenpanzer hinein, was nach dem Optimierungsschritt nicht mehr der Fall ist. Außerdem verbessert sich das Patch, das die aus Sicht der Figur rechte Hand darstellt.

Zusammenfassend lässt sich festhalten, dass das EvOpSeg-Verfahren bei Modellen, die klare konkave Bereiche zwischen den einzelnen Komponenten haben, üblicherweise sehr gute Segmentierungsergebnisse liefert. Typische Beispiele für solche Modelle sind Ameisen und Kraken, aber auch Tische und Bolzen. Darüber hinaus kann man mit dem

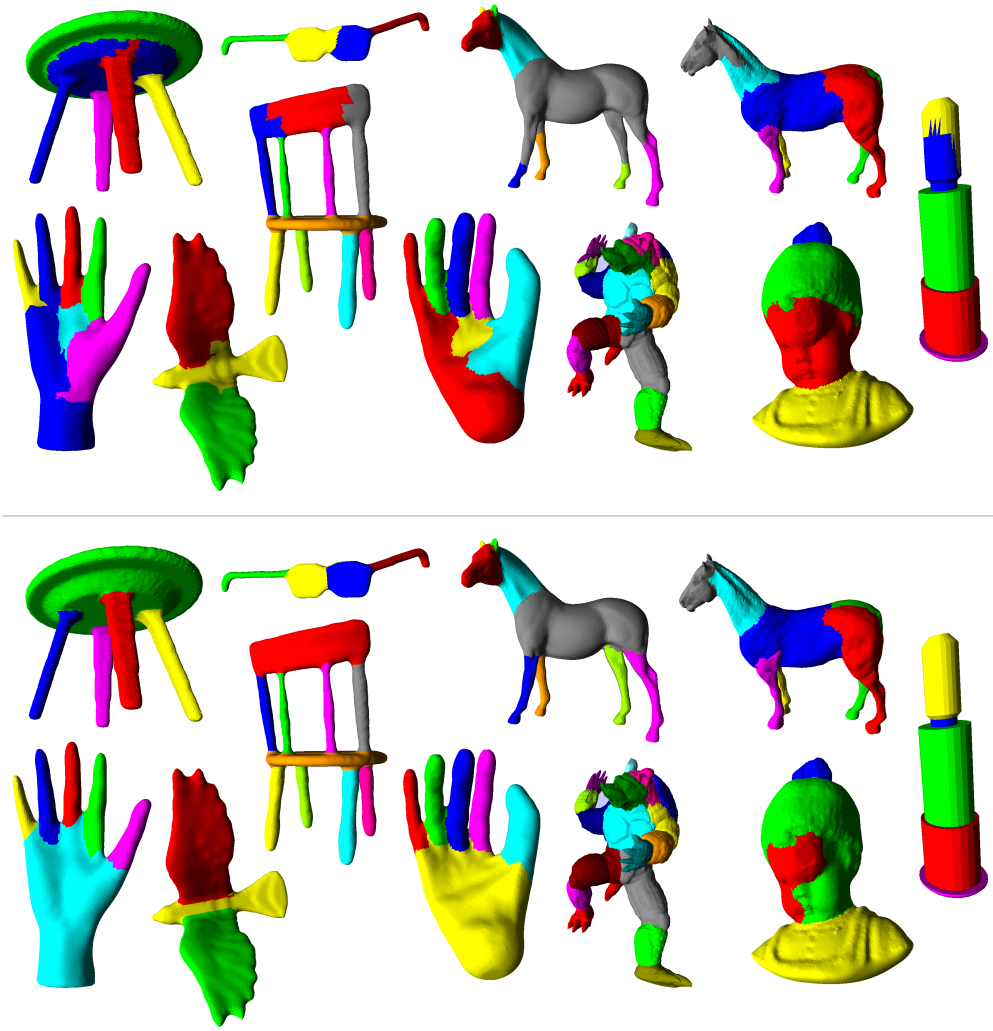


Abbildung 8.19: *Auswirkungen der evolutionären Patch-Optimierung (oben ohne Optimierung, unten mit Optimierung).*

EvOpSeg-Verfahren nicht zuletzt durch die evolutionäre Patch-Optimierung sowie interaktive Eingriffe auch bei „natürlicheren“ bzw. komplexeren Modellen häufig gute oder sogar hervorragende Ergebnisse erhalten.

8.7 Analyse der Optimierungszeit

Ein Nachteil evolutionärer Optimierungsverfahren ist die häufig verhältnismäßig lange Rechenzeit, die zur Ermittlung eines zufriedenstellenden Ergebnisses benötigt wird. Je aufwändiger die Berechnung des Fitnesswertes ist, umso deutlicher ist dieser Nachteil zu spüren. Beim EvOpSeg-Verfahren müssen zur Fitnessberechnung stets für jedes Offspring-Individuum die zugehörigen Patches neu berechnet werden (vgl. auch Listing 5.1). Dies ist im Vergleich zu den anderen Optimierungsschritten wie Rekombinati-

on und Mutation mit Abstand der dominierende Faktor. Eine parallele Berechnung der Fitness für die einzelnen Individuen einer Generation entsprechend Unterabschnitt 7.4.2 bietet daher viel Potential zur Einsparung von Rechenzeit. Im Folgenden wird die zur Patch-Optimierung benötigte Rechenzeit genauer analysiert.

Einige etablierte Verfahren benötigen lediglich wenige Sekunden [LHMR08, AFS06, KLT05, SSCO08], andere wie das in [GF08] vorgestellte hingegen mehrere Minuten. Manche Verfahren wie das auf „Labeling and Learning“ basierende aus [KHS10] setzen voraus, dass im Vorfeld ein Lernprozess anhand bereits segmentierter Modelle stattgefunden hat, die zur gleichen Kategorie gehören wie das zu segmentierende Modell. Eine solche Lernphase kann durchaus mehrere Stunden an Zeit in Anspruch nehmen, führt aber in der Regel dazu, dass für den eigentlichen Segmentierungsvorgang deutlich weniger Zeit benötigt wird. Kalogerakis et al. geben entsprechend an, dass für manche Modelle eine rund achtstündige Lernphase notwendig ist, während die Berechnung der Segmente anschließend nur wenige Minuten dauert [KHS10].

Die zur Patch-Optimierung benötigte Rechenzeit wird hier zunächst anhand des Evaluationslaufs aus Abschnitt 8.4 für das Standard-EvOpSeg-Verfahren unter Verwendung des simulierten manuellen Seeding mit drei Satelliten-Seeds analysiert. Abbildung 8.20 veranschaulicht, für wieviel Prozent der 380 Modelle eine gewisse Anzahl an Minuten zur Patch-Optimierung ausgereicht hat[9]. Für über 46% der Modelle wurde jeweils maximal eine Minute Optimierungszeit benötigt und nur gut 20% der Modelle nahmen mehr als zwei Minuten in Anspruch. Die durchschnittlich für eine Patch-Optimierung benötigte Rechenzeit beträgt rund 103 Sekunden, während der Median der Optimierungszeiten bei 64 Sekunden liegt. Diese Rechenzeiten wurden auf einem Intel Core-i7-Prozessor 2600K mit einer Taktfrequenz von 3.4 GHz und 8 GB RAM ermittelt. Die Fitnesswerte der Individuen einer Generation wurden dabei parallel auf den acht logischen Prozessoren berechnet[10].

Die genannten Angaben zur Optimierungsdauer beziehen sich auf Seeding-Variante 2. Bei Seeding-Variante 1 weichen die Werte für einige Modelle etwas von diesen ab, was in erster Linie damit zusammenhängt, dass dort aufgrund der Mittelung der Anzahl von Ground-Truth-Segmenten eine andere Initial-Seed-Anzahl vorliegt als beim simulierten manuellen Seeding. Bei gleicher Seed-Anzahl sollten die Werte hingegen recht ähnlich sein.

In Abbildung 8.21 ist in Form eines Histogramms dargestellt, für wie viele Patch-Optimierungen jeweils eine bestimmte Anzahl an Sekunden benötigt worden ist. Im linken Teilbild ist die x-Achse linear skaliert, im rechten logarithmisch. Auch hier ist zu erkennen, dass ein Großteil unterhalb von 100 Sekunden liegt; bei einzelnen Modellen ist die Optimierungszeit sogar geringer als 10 Sekunden.

[9] Es wurde jeweils für eine ganzzahlige Anzahl an Minuten ermittelt, wie viele Modelle in maximal dieser Zeit bearbeitet worden sind, und die entsprechenden Punkte in Abbildung 8.20 durch einen Streckenzug interpoliert.

[10] Der Prozessor verfügt über vier echte Prozessor-Kerne. Über einen Hyperthreading-Ansatz stehen jedoch insgesamt acht so genannte *logische Prozessoren* zur Verfügung.

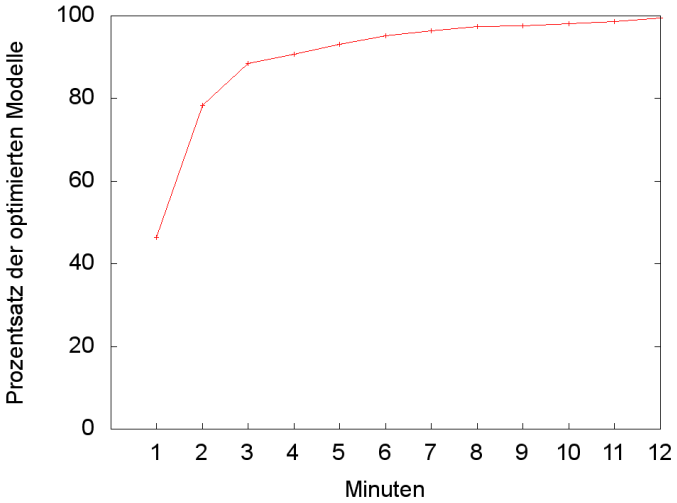


Abbildung 8.20: *Analyse der Optimierungszeit bei drei Satelliten-Seeds.*

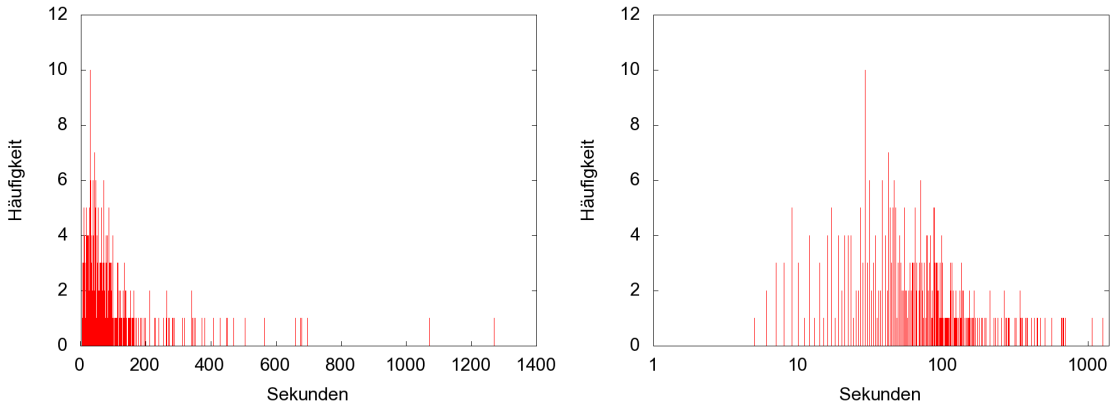


Abbildung 8.21: *Häufigkeiten des Auftretens von Optimierungszeiten als Histogramm (links: lineare Skalierung der x-Achse, rechts: logarithmische Skalierung).*

Verglichen mit einigen anderen Segmentierungsverfahren, deren Rechenzeit im Sekundenbereich liegt, ist das EvOpSeg-Verfahren durch die evolutionäre Patch-Optimierung etwas langsamer. Weitere etablierte Verfahren haben jedoch ebenfalls Laufzeiten im Minutenbereich [GF08, KHS10]. Ein explizites Erlernen von Parametern für einzelne Modellkategorien, was einen zusätzlichen Zeitaufwand für die Trainingsphase bedeutet, ist beim EvOpSeg-Verfahren nicht notwendig. Gleichwohl könnte dies die Qualität der Segmentierungsergebnisse noch weiter steigern.

Bei den genannten Ergebnissen des EvOpSeg-Verfahrens sind acht logische Prozessoren zum Einsatz gekommen. Mit noch mehr Prozessoren bzw. Kernen ist ein noch größerer Speedup, d.h. eine weitere Reduktion der Optimierungszeit zu erwarten. Der *Speedup* ist definiert als $s_n := \frac{t_1}{t_n}$, wobei t_1 die Zeitdauer angibt, die für eine rein sequentielle Abarbeitung auf einem einzigen Prozessor bzw. Prozessor-Kern anfällt, und t_n entsprechend für die bei n Prozessoren bzw. Kernen benötigte Zeit steht [HL09, S. 13]. Bereits im Jahr 1967 merkte Amdahl an, dass der in der Praxis erzielbare Speedup durch den Aufwand für das Verwalten der Daten nach oben begrenzt ist [Amd67]. Eine daraus resultierende Aussage ist als *Amdahl'sches Gesetz* bekannt. Sie besagt, dass der maximal mögliche Speedup durch $\frac{1}{\sigma}$ begrenzt ist, wobei in diesem Zusammenhang σ den Anteil des Aufwandes für die sequentiellen Programmteile an dem Gesamtaufwand bezeichnet [HL09, S. 15].

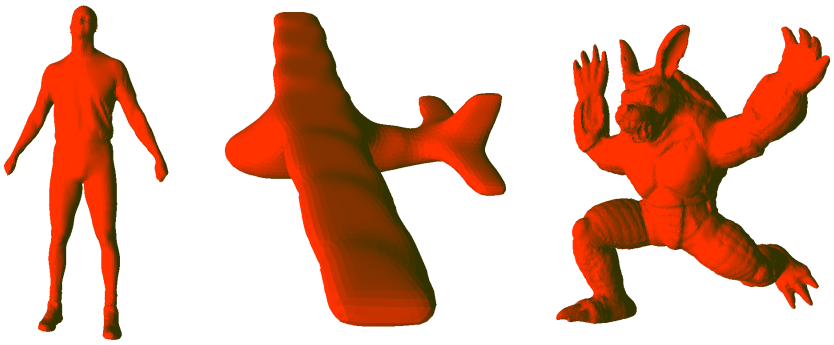


Abbildung 8.22: *Für die Speedup-Untersuchung verwendete Modelle.*

Zur exemplarischen Untersuchung des Potentials der Parallelisierung beim EvOpSeg-Verfahren sind drei Modelle aus dem Princeton-Benchmark herangezogen worden, die in Abbildung 8.22 zu sehen sind. Sie wurden so gewählt, dass sowohl ein Modell mit eher wenigen Dreieckfacetten als auch eines mit recht vielen darunter ist. Anhand dieser Modelle wurde zunächst die für die Optimierung benötigte Rechenzeit in Abhängigkeit von der Anzahl der verwendeten logischen Prozessoren ermittelt. Zum Einsatz kam eine (3,5,15)-ES, die nach jeweils 30 Generationen automatisch beendet worden ist. Bei allen diesen Tests wurden stets die gleichen Seed-Punkte verwendet. In Abbildung 8.23 sind die Ergebnisse zu sehen, links als Angabe der Summe der Optimierungszeiten für die drei Modelle, rechts als entsprechende Speedup-Kurve. Auf den Kurven sind die Werte abgetragen, die sich auf einem Core-i7-2600K-Prozessor, der über acht logische Prozessoren verfügt, ergeben haben. Zusätzlich sind die Ergebnisse, die sich auf einer

Fujitsu Celsius R920 Power-Workstation ergeben haben, als separate Markierungen eingetragen. Diese besitzt zwei physikalische Intel Xeon E5-2690-CPUs mit 2.9 GHz Taktfrequenz und jeweils 8 physikalischen Prozessor-Kernen. Durch Hyperthreading stehen damit 32 logische Prozessoren zur Verfügung. Anhand der beiden Diagramme aus Abbildung 8.23 wird der positive Effekt mehrerer logischer Prozessoren auf die Rechenzeit ersichtlich. Der Anteil der sequentiellen Programmteile beschränkt jedoch den prinzipiell möglichen Speedup, so dass die Speedup-Kurve nicht linear ansteigt.

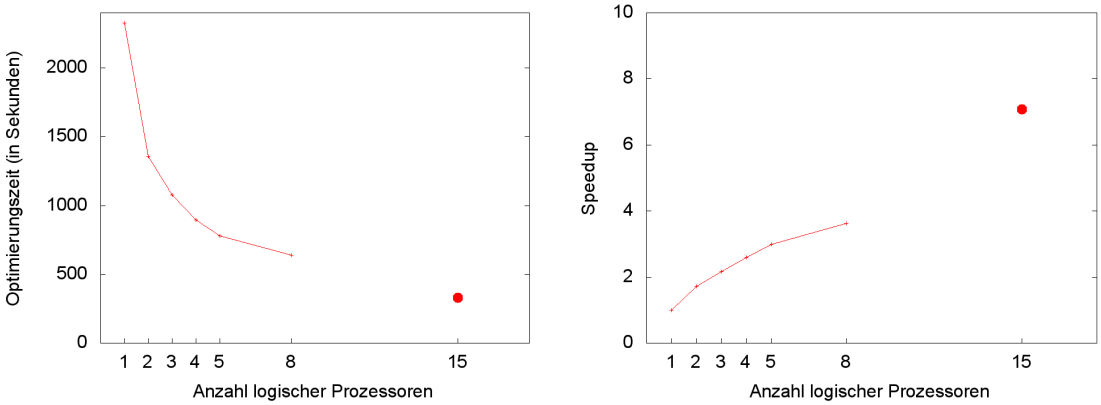


Abbildung 8.23: *Optimierungszeit und Speedup in Abhängigkeit der Anzahl verwendeter logischer Prozessoren.*

In Abbildung 8.24 ist das Ergebnis einer Untersuchung des Zusammenhangs zwischen der Offspring-Anzahl pro Generation und der benötigten kumulierte Rechenzeit für die drei Modelle aus Abbildung 8.22 dargestellt. Die Optimierung erfolgte auch hier mit einer (3,5,15)-ES über 30 Generationen. Anhand der Core-i7-Kurve ist zu erkennen, dass eine Vergrößerung der Offspring-Menge lediglich eine moderate Steigerung der benötigten Rechenzeit nach sich zieht, sofern mindestens so viele logische Prozessoren zur Verfügung stehen wie Offspring-Individuen.

Nach den in diesem Abschnitt gezeigten Ergebnissen sinkt die benötigte Rechenzeit offensichtlich mit steigender Leistungsfähigkeit der Hardware. Eine Vergrößerung der Anzahl logischer Prozessoren kann alternativ jedoch auch dafür verwendet werden, bei gleichbleibender Optimierungsdauer mehr Individuen zu betrachten, um auf diese Weise mitunter noch bessere Lösungen zu erhalten.

8.8 Fazit

Die in dieser Arbeit verwendeten Metriken *Rand-Index* und *Konsistenzfehler* haben sich im Bereich der Segmentierung von Dreiecksnetzen zur Bestimmung der Qualität von Segmentierungen etabliert. Mit ihnen lassen sich Segmentierungsverfahren quantitativ miteinander vergleichen. Eine Betrachtung dieser Metriken innerhalb einer entsprechenden Evaluation anhand des recht umfangreichen Princeton-Benchmark ergab,

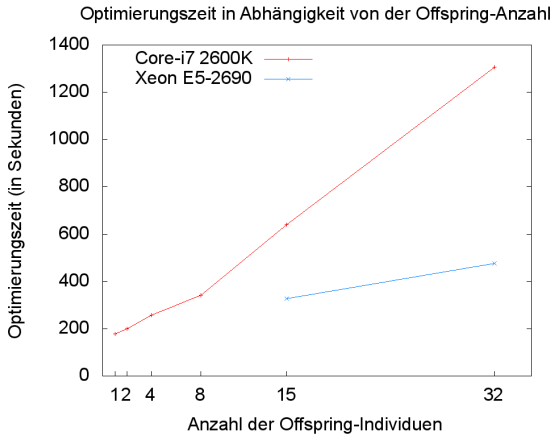


Abbildung 8.24: *Optimierungszeit in Abhängigkeit von der Anzahl der Offspring-Individuen.*

dass mit dem EvOpSeg-Ansatz insbesondere für eine manuelle und simuliert-manuelle Seed-Vergabe im Vergleich zu anderen etablierten Verfahren gute Segmentierungsergebnisse zu erzielen sind. Dabei sind die „manuellen Seeds“ von mehreren Testpersonen mit einem speziell dafür entwickelten Werkzeug gesetzt worden. Obwohl viele Testpersonen zuvor noch keinerlei Berührungspunkte mit Programmen der dreidimensionalen Computergrafik sammeln konnten, waren die Ergebnisse im Mittel überzeugend. Die in dieser Arbeit genutzte Variante zur automatischen Seed-Anordnung schneidet hingegen etwas schlechter ab. Dennoch ist das EvOpSeg-Verfahren auch dann noch einigen der etablierten Verfahren überlegen, wenn automatisch gesetzte Initial-Seeds verwendet werden. Insbesondere bei Modellen mit vielen längeren, klar erkennbaren Extremitäten erweist sich diese Seeding-Variante als durchaus geeignet.

Anhand zweier Ansätze zum Vergleich von Seed-Konfigurationen konnte gezeigt werden, dass das simulierte manuelle Seeding offensichtlich ein echtes manuelles Seeding, bei dem die Initial-Seeds mittig auf den wahrgenommenen Komponenten angeordnet werden, recht gut approximiert. Auch die damit erreichten Rand-Index- und Konsistenzfehler-Werte entsprechen ziemlich genau denen, die bei Verwendung der manuell gesetzten Seeds erreicht werden. Dabei haben die Anwendenden jedoch keineswegs interaktiv gehandelt, d.h. es sind keine Patches angezeigt worden, so dass die Personen nicht darauf reagieren konnten. Wird nicht das `Seeding-Tool` sondern das `EvOpSeg-Tool` verwendet, so lassen sich mit interaktiven Eingriffen noch deutlich bessere Segmentierungen erzielen.

Technische Modelle wie solche aus dem CAD-Bereich weisen oft starke konkave Regionen in den Bereichen zwischen zwei Komponenten auf. Bei solchen Modellen zeigt das EvOpSeg-Verfahren normalerweise keine Schwächen. Auch bei Modellen, die klar erkennbare Extremitäten haben (wie die in dieser Arbeit vorkommenden Ameisen- oder Krakenmodelle) ergeben sich meist sehr gute Segmente. Etwas schwieriger sind hingegen die Modellkategorien „Armadillo“, „Büsten“ und „Menschen“. Hier sind zwar stets zumindest einige der ermittelten Segmente als „gut“ anzusehen, oft gibt es jedoch bei

anderen Segmenten auch einige suboptimale Grenzen, wenn pro Komponente nur ein Initial-Seed verwendet wird.

Der Zeitaufwand für das Platzieren von Initial-Seeds ist sicherlich nicht signifikant größer als der für die Bedienung anderer interaktiver Programme zur Ermittlung von 3D-Segmentierungen. Auch die Berechnung einer einzelnen Segmentierung nimmt üblicherweise nur einige Bruchteile einer Sekunde bis hin zu wenigen Sekunden in Anspruch. Der Mehrwert, den die evolutionäre Patch-Optimierung liefert, wird mit einer etwas längeren Rechenzeit erkauft. Allerdings wurden bei der in diesem Kapitel beschriebenen Auswertung pro Modell im Schnitt lediglich 103 Sekunden für die Optimierung benötigt, für 80% der Modelle konnten innerhalb von zwei Minuten optimierte Segmente berechnet werden.

Kapitel 9

Zusammenfassung und Ausblick

Im Bereich der 3D-Computergrafik gibt es zahlreiche Anwendungsgebiete, für die eine möglichst gute Zerlegung dreidimensionaler Modelle in Segmente wichtig ist. Als Beispiele dafür seien die Bereiche Modellierung, Texture Mapping, 3D-Shape Retrieval, Mesh Compression oder Skeleton Extraction genannt. Ziel der vorliegenden Arbeit war die Entwicklung eines Segmentierungsverfahrens, mit dem man sowohl technische als auch natürliche Modelle in bedeutungsvolle Segmente zerlegen kann. Dabei sollten Interaktionen möglich sein, ohne dass jedoch zu viele Interaktionstätigkeiten zwingend erforderlich sind. Ein auf Seed-Punkten basierender Ansatz scheint somit ideal zu sein, da Seeds sehr einfach manuell gesetzt sowie verschoben werden können und jedes Segment bzw. Patch nur einen oder aber sehr wenige Seeds erfordert.

Neben der Möglichkeit manueller Eingriffe eignet sich ein Seed-basierter Ansatz darüber hinaus auch gut für die Optimierung der Patches mit einem Evolutionären Algorithmus. Ein solcher Optimierungsansatz ist zentraler Bestandteil des in dieser Arbeit vorgestellten EvOpSeg-Verfahrens. Um bessere Steuerungsmöglichkeiten – sowohl für manuelle Interaktionen als auch für die evolutionäre Patch-Optimierung – zu erzielen, wurde zudem das Prinzip des Satelliten-Seeding eingeführt.

Die sich aus dem EvOpSeg-Ansatz ergebenden zentralen Ergebnisse sowie die Erkenntnisse der Arbeit werden im folgenden Abschnitt kurz zusammengefasst, bevor in Abschnitt 9.2 eine Diskussion einiger weiterführender, über das Thema dieser Arbeit hinausgehender Fragestellungen erfolgt.

9.1 Ergebnisse der Arbeit

Beim EvOpSeg-Verfahren erfolgt die Berechnung der Patches ausgehend von den auf der Oberfläche angeordneten Seed-Punkten. Für die Zuordnung der Netzdreiecke zu den einzelnen Seeds sind in dieser Arbeit spezielle Distanzfunktionen definiert worden, die neben der geodätischen Distanz auf der Oberfläche des Modells auch die Krümmung sowie gegebenenfalls lokale Formmerkmale berücksichtigen. Die Einbeziehung von Formmerkmalen hat sich bei der Patch-Berechnung manchmal als vorteilhaft

und manchmal als nachteilig herausgestellt. Aus diesem Grund wird für das jeweilige Modell innerhalb der evolutionären Patch-Optimierung unter anderem auch ermittelt, ob die Berücksichtigung solcher Merkmale bei der Distanzberechnung das Segmentierungsergebnis positiv beeinflusst.

Anhand mehrerer Beispiele wurde gezeigt, dass die alleinige Verwendung von Initial-Seeds häufig nicht zu den gewünschten Segmentierungsergebnissen führt. Die Initial-Seeds bieten nämlich im Allgemeinen nicht genügend Freiheitsgrade, um durch Verschieben einiger Seeds oder durch Anpassen der Seed-Gewichte zufriedenstellende Segmente zu erhalten. Dabei spielt es keine Rolle, ob die genannten Veränderungen manuell oder durch die evolutionäre Patch-Optimierung erfolgen. Aufgrund der verwendeten Zielfunktion, die nicht nur bewertet, wie konkav die Bereiche um die Patch-Grenzen herum sind, sondern darüber hinaus auch lange Patch-Grenzen „bestraft“, tendiert die evolutionäre Patch-Optimierung bei ausschließlicher Verwendung von Initial-Seeds sogar dazu, einige deutlich zu klein ausfallende Patches zu erzeugen. Das neu eingeführte Konzept der Satelliten-Seeds sorgt hier für Abhilfe. Satelliten-Seeds reduzieren jedoch nicht nur die Wahrscheinlichkeit für das Auftreten zu kleiner Patches, sondern sie ermöglichen darüber hinaus auch die gezielte Beeinflussung eines Teilstücks des Patch-Randes, ohne dabei zwangsläufig die übrigen Stücke des Randes zu verändern.

Einige etablierte Segmentierungsverfahren nutzen eine Boundary-Smoothing-Technik innerhalb eines Nachbearbeitungsschrittes, um die berechneten Segmentgrenzen zu glätten. Während dadurch nur sehr lokal Einfluss auf die Grenzen genommen wird, ermöglicht die evolutionäre Patch-Optimierung sogar eine großflächige Beeinflussung der Patches. Ein damit verbundener positiver Effekt konnte bei zahlreichen Modellen beobachtet werden. Bei manchen Modellen ist eine evolutionäre Patch-Optimierung jedoch gar nicht erst notwendig, da aufgrund der verwendeten Distanzfunktion bereits die Patches der Initialsegmentierung als „perfekt“ zu bezeichnen sind.

Die Evaluation der Segmentierungsergebnisse erfolgte anhand eines umfangreichen Benchmark. Mit Hilfe der Metriken Rand-Index und Konsistenzfehler lassen sich die EvOpSeg-Ergebnisse recht objektiv mit denen von etablierten Segmentierungsansätzen vergleichen. Neben dem Standard-EvOpSeg-Verfahren sind auch die beiden Varianten der gebundenen Mutation bzw. der zweistufigen Optimierung ausgewertet worden. Die Ergebnisse des Standard-EvOpSeg-Verfahrens scheinen insgesamt etwas besser zu sein als die der beiden anderen Varianten. Allerdings hat sich die gebundene Mutation dann bewährt, wenn Initial-Seeds recht nah an der Grenze logischer Komponenten liegen. Dies kann insbesondere beim automatischen Seeding vorkommen. Obwohl sich diese Seeding-Variante bei zahlreichen Modellen insofern als gut erweist, als dass auf jeder Komponente mindestens ein Initial-Seed liegt, ist dies längst nicht bei allen Modellen der Fall. Gerade dann, wenn einige der Komponenten im Vergleich zu anderen eher klein sind, erhalten diese bei der automatischen Variante häufig keine Initial-Seeds, während zugleich auf den großen Komponenten mehrere Seeds in einer etwas wahllos wirkenden Weise verteilt sind.

Bei Verwendung manuell gesetzter Seeds stehen die drei oben genannten EvOpSeg-Varianten den etablierten Verfahren bezogen auf die beiden Fehler-Metriken in nichts

nach, sondern sind ihnen vielmehr – gemittelt über alle Modelle – überlegen. Dabei sind die Seeds keineswegs von Experten auf den Modellen angeordnet worden, sondern von mehreren Testpersonen mit ganz unterschiedlichen beruflichen Hintergründen; insbesondere hatten viele Testpersonen zuvor gar keine Berührungspunkte mit dem Bereich der dreidimensionalen Computergrafik, und erst recht nicht mit dem `EvOpSeg-Tool`. Außerdem ist zu berücksichtigen, dass die Anordnung der Seeds ohne jegliches Feedback erfolgte, d.h. den Testpersonen wurden keine Patches präsentiert. Eine solche Anzeige von Patches während der manuellen Anordnung von Seeds stellt den ersten Schritt einer interaktiven Segmentierung dar. Anhand einer Teilmenge der Benchmark-Modelle wurde gezeigt, dass ein interaktives Vorgehen die Qualität der Segmente sogar noch weiter verbessern kann.

Eine manuelle Anordnung von Initial-Seeds auf einem „durchschnittlichen“ dreidimensionalen Modell kann prinzipiell innerhalb von rund 100 Sekunden erfolgen. Bei einem einzelnen Modell ist die für diese Interaktionsart benötigte Zeit also vergleichsweise gering. Anders sieht es hingegen aus, wenn eine große Anzahl von Modellen zu bearbeiten ist. In einem solchen Fall muss außerdem mit einer Ermüdung der Person, die die Initial-Seeds setzt, gerechnet werden. Dies wirkt sich oft negativ auf die Qualität aus. Beispielsweise kann eine Testperson nach längerer Dauer vergessen, einzelne Komponenten mit Initial-Seeds zu versehen, womit dann auch die entsprechenden Segmente fehlen. Aufgrund derartiger Ermüdungseffekte sowie der Tatsache, dass für eine benchmarkbasierte Auswertung zahlreiche Testpersonen benötigt werden, ist ein für Mesh-Segmentation-Benchmarks geeignetes Verfahren zur Simulation einer manuellen Seed-Vergabe entwickelt worden. Die Bewertungen der daraus resultierenden Segmentierungsergebnisse entsprechen weitgehend denen, die sich bei einer manuellen Seed-Vergabe durch Testpersonen ergeben, und sind damit besser als die von vergleichbaren etablierten Segmentierungsverfahren. Eine Untersuchung aller bei der Evaluation betrachteten Seed-Konfigurationen hat ergeben, dass das simulierte manuelle Seeding einem echten manuellen tatsächlich sehr ähnlich ist, während die rein automatische Seed-Vergabe von beiden stärker abweicht.

9.2 Weiterführende Fragestellungen

Der in der vorliegenden Dissertation präsentierte EvOpSeg-Ansatz befasst sich mit der Segmentierung dreidimensionaler Modelle. Sind mehrere Modelle der gleichen oder ähnlicher Kategorien zu segmentieren, geschieht dies momentan vollkommen unabhängig voneinander. Möglicherweise kann jedoch in solchen Situationen eine abgewandelte Variante des EvOpSeg-Verfahrens, bei der eine modifizierte Patch-Optimierung auf allen oder zumindest einigen der Modelle gleichzeitig stattfindet, die Segmentierungsqualität noch weiter steigern. Auf die Weise kann versucht werden, die Ränder korrespondierender Segmente auf den unterschiedlichen Modellen in etwa gleich verlaufen zu lassen, so dass die Segmente anschließend recht einfach gegeneinander austauschbar sind, was insbesondere für den Bereich der Modellierung interessant ist. Dazu müssen die Seeds aller betrachteten Modelle mit geeigneten Labels versehen werden. Zu untersuchen ist, mit

welcher konkreten Anpassung der Zielfunktion sich modellübergreifend ähnliche Segmenträndern erzielen lassen. Möglicherweise kann die Zielfunktion aus Gleichung (7.9), in der $h_{concave}$ und h_{length} miteinander verknüpft sind, um einen geeigneten dritten Term erweitert werden, der angibt, wie ähnlich korrespondierende Segmenträndern einander auf unterschiedlichen Modellen sind. Die Bewertung der Ähnlichkeit ist an dieser Stelle sicherlich der entscheidende Punkt.

Als ein positiver Effekt des Satelliten-Seeding hat sich herausgestellt, dass diese Seeds häufig die evolutionäre Patch-Optimierung daran hindern, „zu kleine“ Segmente zu erzeugen. Bei sehr detaillierten Modellen klappt dies jedoch nicht immer zuverlässig. So war gelegentlich bei Armadillomodellen festzustellen, dass ein einzelner Finger ein eigenständiges Segment ergibt, während der Rest der Hand dem Arm zugeordnet ist. Sinnvoll ist hingegen, statt des Fingers die komplette Hand als ein Segment zu betrachten. Hier bietet sich eine weitergehende Untersuchung an, ob man für spezielle Anwendungsdomänen bzw. Modellkategorien konkrete Kriterien vorgeben kann, die bei der evolutionären Patch-Optimierung zu beachten sind. Ein solches Kriterium könnte beispielsweise sein, dass die Segmentgrößen nicht zu stark voneinander abweichen dürfen[1]. Auch dafür ist eine geeignete Anpassung der Zielfunktion notwendig.

Bislang ungeklärt ist auch die Frage, ob man die zu einer optimierten Segmentierung gehörenden Seeds auf ähnliche, noch zu segmentierende Modelle übertragen kann. Dies könnte nicht nur das Seeding erleichtern, sondern mitunter auch die benötigte Optimierungszeit reduzieren. Eventuell kann aber auch eine Übertragung der gesamten Segmente mit Hilfe eines Morphing-Ansatzes erfolgen. In dem Fall ist jedoch damit zu rechnen, dass die Segment-Ränder auf dem neuen Modell nicht zufriedenstellend sind, so dass auch hier eine Optimierung notwendig ist. Zur Generierung geeigneter Initial-Seeds könnte das simulierte manuelle Seeding aus Kapitel 8 in Kombination mit den auf das neue Modell übertragenen Segmenten genutzt werden. Ob dies einen messbaren Mehrwert liefert, bleibt zu untersuchen.

Beim EvOpSeg-Ansatz werden Größe und Form der Netzdreiecke nicht gesondert berücksichtigt. Einige Vernetzungsprogramme führen bei der Konstruktion eines Dreiecksnetzes jedoch dazu, dass ebene Abschnitte des Modells mit wenigen, eher großen Dreiecken realisiert werden und zylinderförmige Regionen aus schmalen, „langgezogenen“ Dreiecken bestehen. In solchen Fällen können aus der Größe und Form der Dreiecke zusätzliche Informationen abgeleitet werden, mit denen sich die Genauigkeit des merkmalbasierten Distanzanteils aus Gleichung (5.28) möglicherweise erhöhen lässt. Die Erzielung eines möglichen Mehrwerts hängt hierbei aber substantiell von der Arbeitsweise des bei der Modell-Konstruktion verwendeten Vernetzungstools ab.

Das in dieser Arbeit verwendete automatische Seeding, welches auch Bestandteil von Seeding-Variante 1 ist, hat sich bei einigen Modellkategorien gut bewährt, während damit bei anderen Kategorien nicht jede semantische Komponente einen Initial-Seed erhält. Auch wenn sich mit verhältnismäßig wenigen Interaktionen in Form einer manuellen Initial-Seed-Vergabe sicherlich bessere Ergebnisse erzielen lassen als mit au-

[1] Kalogerakis et al. berücksichtigen bei ihrem *Segment-Weighted Error* ebenfalls die Größe der einzelnen Segmente [KHS10].

tomatisch angeordneten Seeds, können weiterführende Forschungsarbeiten durchaus die Entwicklung geeigneter kategoriespezifischer Strategien zur automatischen Seed-Vergabe zum Thema haben.

Abschnitt 8.5 hat sich mit einer manuellen Anordnung von Initial-Seeds durch Testpersonen befasst. Zentrale Rolle spielte in diesem Zusammenhang ein so genanntes *Seeding ohne Feedback*, bei dem keine Patches zu sehen sind. Jede Testperson hat auf diese Weise 19 dreidimensionale Modelle mit Initial-Seeds versehen, so dass insgesamt auf allen 380 Benchmark-Modellen Initial-Seeds angeordnet worden sind. In diesem Zusammenhang gemachte Beobachtungen lassen vermuten, dass einige Personen dazu tendieren, mit fortschreitender Anzahl bearbeiteter Modelle Seeds weniger akribisch zu setzen als bei den ersten Modellen, was auf einen gewissen Ermüdungseffekt hindeuten kann. Ob dies wirklich der Fall ist, kann zukünftig – genauso wie der konkrete Einfluss von Ermüdungserscheinungen auf die resultierende Segmentierungsqualität – genauer untersucht werden. Dazu könnten beispielsweise mehrere Testpersonen dieselben Modelle mit Initial-Seeds versehen, wobei die Modelle dann aber in unterschiedlichen Reihenfolgen präsentiert werden.

Anhang A

Software und Dateiformat

Die Darstellung der Modelle erfolgt bei den beiden im Rahmen dieser Arbeit entstandenen Programmen `EvOpSeg-Tool` und `Seeding-Tool` mit Hilfe von DirectX 9. Verwendet wurde die Programmiersprache C++ in Kombination mit dem Framework wxWidgets 2.8.9. Beide entwickelten Programme verarbeiten dreidimensionale Modelle, die im PLY-Dateiformat vorliegen. Das `EvOpSeg-Tool` und das `Seeding-Tool` werden in den Abschnitten A.1 bzw. A.2 vorgestellt, wobei der Fokus auf den für diese Arbeit wesentlichen Funktionalitäten der Programme liegt. Abschnitt A.3 geht anschließend auf das PLY-Dateiformat ein.

A.1 EvOpSeg-Tool

Das `EvOpSeg-Tool` ist als Multi-Window-Programm realisiert worden. Da die Größen der einzelnen Fenster unabhängig voneinander änderbar sind, kann eine individuell auf die zur Verfügung stehende Bildschirmauflösung abgestimmte Anordnung erfolgen. Die Fenster des `EvOpSeg-Tool` sind in Abbildung A.1 dargestellt. Das Modell wird in einem so genannten *Modellfenster* angezeigt. In diesem Fenster können beispielsweise einzelne Seeds manuell gesetzt oder Patches selektiert werden. Grundlegende Segmentierungsfunktionalitäten werden hingegen über ein *Steuerungsfenster* zur Verfügung gestellt. Zwei optionale Fenster können über das Menü `Fenster` des Steuerungsfensters ein- und ausgeblendet werden: ein *Nachrichtenfenster* zur Ausgabe von Log-Meldungen und ein *Fitnessfenster* zur Anzeige des Fitnessverlaufs bei der Patch-Optimierung.

Steuerung

Für die Anzeige der Modelle ist ein Flat-Shading gewählt worden, damit die einzelnen Dreiecke besser erkennbar und somit Seed-Dreiecke einfacher zu selektieren sind. Dadurch sind auch die Grenzen zwischen den einzelnen Segmenten deutlicher sichtbar als beispielsweise beim Phong-Shading. Das manuelle Drehen eines Modells erfolgt durch

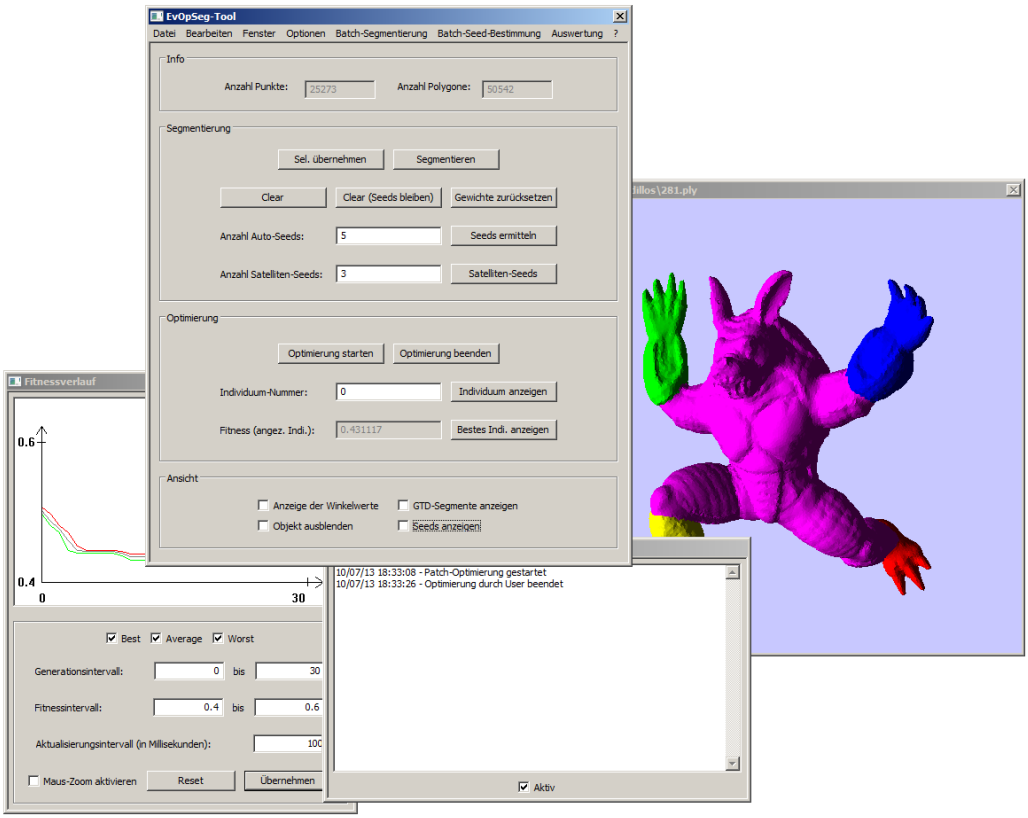


Abbildung A.1: *Die vier Fenster des* `EvOpSeg-Tool`.

einen Linksklick in das Modellfenster und Bewegung der Maus bei gedrückt gehaltener Maustaste. Gezoomt wird durch Drehung des Mausrads oder alternativ durch Cursor-Tasten („aufwärts“ bzw. „abwärts“). Ein schnelleres Hinein- bzw. Herauszoomen ist möglich, indem beim Drehen des Mausrads die `[STRG]`-Taste betätigt wird.

Seeding und Segmentierung

Durch Drücken des Button `Seeds ermitteln` werden Initial-Seeds entsprechend dem automatischen Seeding aus Unterabschnitt 5.1.2 auf dem Modell angeordnet. Die gewünschte Anzahl wird in dem links vom Button befindlichen Eingabefeld angegeben. Bei dieser automatischen Ermittlung von Initial-Seeds werden alle zuvor vorhandenen Seeds gelöscht und durch die neu ermittelten ersetzt. Sollen hingegen zusätzlich zu vorhandenen Seeds weitere Initial-Seeds automatisch angeordnet werden, was beispielsweise beim hybriden Initial-Seeding der Fall sein kann, ist der Menü-Eintrag `Bearbeiten` ▷ `Additive Auto-Seeds` zu wählen. Auch dabei wird die Anzahl der hinzuzufügenden Initial-Seeds aus dem zuvor erwähnten Eingabefeld übernommen.

Um manuell Seeds auf dem Modell anzuordnen, wird das gewünschte Dreieck des Modells bei gedrückter `[STRG]`-Taste per Linksklick selektiert. Diese Selektion stellt allerdings zunächst noch keinen Seed dar. Sagt die Lage den Anwendenden nicht zu,

können sie direkt eine neue Stelle selektieren – die neue Selektion ersetzt dann die vorherige. Soll ein selektiertes Dreieck als Seed übernommen werden, ist der Button `Sel. übernehmen` des Steuerungsfensters zu betätigen.

Satelliten-Seeds werden durch Betätigung des gleichnamigen Button hinzugefügt. Wie bei den Initial-Seeds kann auch hier die gewünschte Anzahl in einem entsprechenden Eingabefeld angegeben werden.

Die Anwendenden haben unterschiedliche Möglichkeiten, Einfluss auf vorhandene Seeds zu nehmen. So können sie Seeds verschieben, indem zunächst ein Seed wie oben beschrieben selektiert wird (Linksklick mit gedrückter `[STRG]`-Taste) und `Bearbeiten` ▷ `Seed verschieben` gewählt wird. Anschließend muss noch die Stelle markiert werden, an die der Seed zu verschieben ist. Im Unterschied zum Löschen von Seeds – was ebenfalls über das Menü `Bearbeiten` oder aber durch `[STRG]`+Rechtsklick geschieht – gefolgt von dem Platzieren eines neuen, behält ein verschobener Seed seinen Gewichtswert. Die Änderung eines Gewichtswertes erfolgt über den Menüpunkt `Bearbeiten` ▷ `Seed-Gewicht ändern`; dazu muss der Seed zuvor selektiert sein. Der Button `Gewichte zurücksetzen` bewirkt, dass alle Seed-Gewichte auf 1.0 gesetzt werden.

Die Berechnung der Patches wird über den Button `Segmentieren` angestoßen. Um die formbasierte Distanz zu nutzen, muss vor dem Drücken dieses Button zunächst die Berechnung der formbasierten Merkmalsvektoren erfolgt sein. Dies geschieht über den Menüpunkt `Bearbeiten` ▷ `Berechnung formbasierter Merkmalsvektoren`. Eine Ausnahme stellen die weiter unten beschriebenen Batch-Läufe dar, bei denen diese Merkmalsvektoren ohne manuelle Interaktionen ermittelt werden.

Gelöscht wird eine Segmentierung über den Button `Clear`. Dabei werden auch sämtliche Seeds entfernt. Der Button `Clear (Seeds bleiben)` sorgt hingegen für das Entfernen der Patches, nicht aber der Seeds. Dies kann unter anderem dann relevant sein, wenn Seed-Gewichte verändert worden sind, da dann nicht immer eine inkrementelle Patch-Aktualisierung ausreicht (vgl. auch Abschnitt 5.3).

Gruppierung

Patches können durch einen Rechtsklick bei gedrückter `[SHIFT]`-Taste selektiert werden. Zwei selektierte Patches lassen sich über `Bearbeiten` ▷ `Gruppieren` zu einem neuen Patch bzw. einer Patch-Gruppe zusammenfassen. Durch den Menüpunkt `Bearbeiten` ▷ `Gruppierung aufheben` wird eine Patch-Gruppe durch alle enthaltenen Patches ersetzt.

Speichern

Ein segmentiertes oder mit Seeds versehenes Modell kann gespeichert und später weiterverarbeitet werden (Menüpunkt `Datei` ▷ `Modell speichern`). Es lassen sich jedoch auch gezielt einzelne Patches bzw. Segmente speichern. Dazu sind die entsprechenden Patches zu selektieren und anschließend `Datei` ▷ `Selektion speichern` zu wählen.

Optimierung

Eine Patch-Optimierung kann über die Button `Optimierung starten` und `Optimierung beenden` gestartet bzw. wieder beendet werden. Bei dieser Variante der Patch-Optimierung gibt es kein Terminierungskriterium im eigentlichen Sinne; stattdessen ist von Anwenderseite her der Vorgang explizit zu beenden. Im Fitnessfenster wird der Verlauf der Fitness in Echtzeit angezeigt. Durch Angabe der Individuen-Nummer kann die Segmentierung angezeigt werden, die zu dem entsprechenden Individuum gehört. Darüber hinaus ist auch die Anzeige des besten Individuums aller Generationen vorgesehen. Aus implementierungstechnischen Gründen kann die Optimierung bedingt durch das Plotten der Fitnesskurven nur auf einem einzigen logischen Prozessor erfolgen. Eine parallele Patch-Optimierung ist hingegen über drei Menüpunkte unterhalb von `Bearbeiten` möglich. Dort wird sowohl die Patch-Optimierung des Standard-EvOpSeg-Verfahrens angeboten, als auch eine Patch-Optimierung mit gebundener Mutation sowie eine zweistufige Patch-Optimierung. Diesen drei liegt als Terminierungskriterium eine vorgegebene Anzahl von Generationen zugrunde, die in dem ES-Einstellungsfenster (`Optionen` ▷ `ES-Einstellungen`) geändert werden kann.

Auswertung

Im Princeton-Mesh-Segmentation-Benchmark sind für jedes Modell durchschnittlich elf Ground-Truth-Segmentierungen enthalten. Ein solcher Ground-Truth-Datensatz lässt sich über `Datei` ▷ `Segmentierungsdaten laden` importieren. Voraussetzung dafür ist, dass das zugehörige Benchmark-Modell aktuell geladen ist. Ob die Ground-Truth-Segmente angezeigt werden oder das Modell in gewohnter Weise dargestellt wird, steuert man über die Checkbox `GTD-Segmente anzeigen`. Ist eine berechnete Segmentierung vorhanden und ein Ground-Truth-Datensatz geladen, werden die zugehörigen Konsistenzfehler bzw. Rand-Index-Werte im Nachrichtenfenster angezeigt, wenn die entsprechenden Punkte im Menü `Auswertung` gewählt werden.

Es ist möglich, für alle 380 Modelle des Benchmark aus [CGF09] Segmentierungen mit dem `EvOpSeg-Tool` in einem Batch-Durchlauf zu ermitteln. Dazu finden sich unter dem Menü `Batch-Segmentierung` die entsprechenden Einträge. Ausgehend von dem aktuell geladenen Modell werden dieses und alle (bezüglich des Dateinamens) chronologisch folgenden behandelt. Dabei müssen jedoch die Dateinamen die Form `i.ply` mit $\texttt{i} \in \{1, \ldots, 400\}$ haben.

Unter anderem für die Auswertung der Qualität von automatischem und simuliert-manuellem Seeding werden (in Dateien gespeicherte) Modelle benötigt, die mit eben solchen Seeds versehen sind. Die Erzeugung solcher Dateien erfolgt analog zur Batch-Segmentierung durch die beiden Einträge des Menüs `Batch-Seed-Bestimmung`.

Unter dem Menü `Auswertung` werden Funktionalitäten zur Ermittlung der beiden in Abschnitt 8.5 genannten Hausdorff-Abstände angeboten. Dies kann wahlweise als Batch-Lauf über mehrere Dateien oder für ein konkretes Modell erfolgen. Im ersten Fall werden die Ergebnisse in einer CSV-Datei im gleichen Verzeichnis gespeichert, im zweiten wird das Ergebnis im Nachrichtenfenster angezeigt.

A.2 Seeding-Tool

Wie schon das `EvOpSeg-Tool` ist auch das `Seeding-Tool` als Multi-Window-Programm realisiert. Hier existieren allerdings lediglich zwei Fenster: ein *Steuerungsfenster* sowie das *Modellfenster*. Diese sind in Abbildung A.2 zu sehen. Im Modellfenster des `Seeding-Tool` ordnen die Anwendenden Seeds auf dem Modell an, ohne sich dabei Patches anzeigen lassen zu können. Neben der Anzeige, wie viele Punkte bzw. Dreiecke das Dreiecksnetz enthält, dient das Steuerungsfenster vor allem dem Laden und Speichern der Modelle sowie der Übernahme der Selektionen als Seeds.

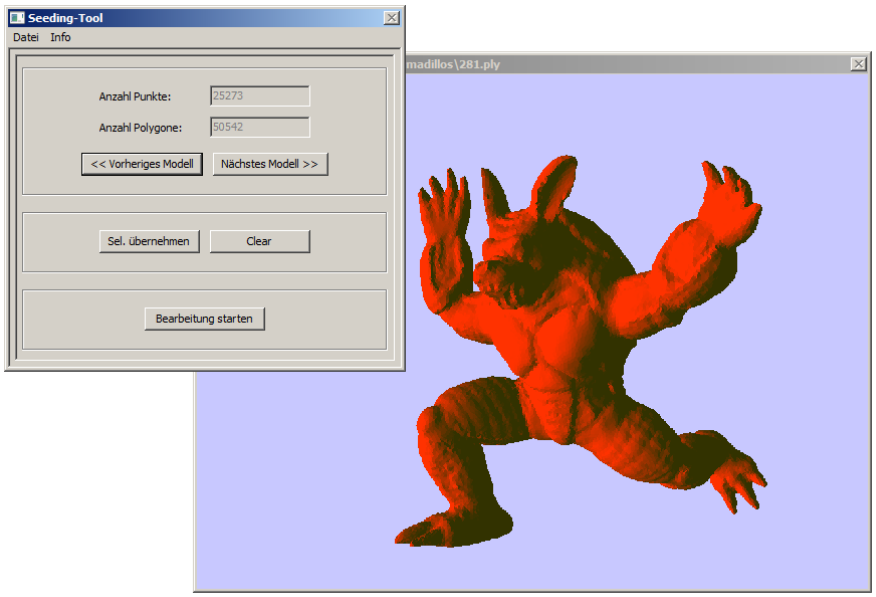


Abbildung A.2: *Steuerungs- und Modellfenster des* `Seeding-Tool`.

Beim Start des Programms ist das als erstes zu bearbeitende Modell auszuwählen. Über den Menüpunkt `Datei` ▷ `Modell laden` lässt sich die Wahl anschließend ggf. korrigieren. Bei der aktuellen Version des Programms müssen die Dateien aller Modelle, die von einer Testperson mit Seeds zu versehen sind, im gleichen Verzeichnis liegen. Zudem muss jeder Dateiname der Form `i.ply` mit $\texttt{i} \in \{1, \ldots, 400\}$ genügen.

Die Steuerung ist der des `EvOpSeg-Tool` recht ähnlich. Ein Dreieck des Modells wird markiert, indem es bei gedrückter `[STRG]`-Taste per linker Maustaste angeklickt wird. Die anschließende Selektion eines anderen Dreiecks hebt eine vorherige Selektion auf. Um ein selektiertes Dreieck als Seed zu markieren, ist der Button `Sel. übernehmen` im Steuerungsfenster zu betätigen. Einzelne Seeds werden gelöscht, indem sie bei gedrückter `[STRG]`-Taste mit der rechten Maustaste angeklickt werden. Das Entfernen sämtlicher Seeds eines Modells erfolgt über den Button `Clear`.

Um Testpersonen eine einfache Bearbeitung mehrerer Modelle zu ermöglichen, enthält das Steuerungsfenster die Buttons `<< Vorheriges Modell` sowie `Nächstes Modell >>`. Bei Betätigung dieser Schaltflächen wird das aktuelle Modell inklusive der manuell

gesetzten Seeds gespeichert und das Modell geladen, dessen PLY-Datei sich im selben Verzeichnis befindet wie die des aktuellen Modells und das zudem bezüglich des Dateinamens das lexikographisch folgende bzw. vorherige Modell ist. Dabei werden nach jedem Ladevorgang auch sämtliche Seeds angezeigt, die bereits zuvor auf dem Modell angeordnet worden sind.

Durch Betätigung des Button `Bearbeitung starten` beginnt eine Zeitmessung, die für jedes Modell ermittelt, wie viele Sekunden die Testperson zur Platzierung von Seeds benötigt. Die Zeit, die für das Speichern und Laden von Modellen benötigt wird, fließt dabei nicht in das Ergebnis mit ein, das in Form einer CSV-Datei im aktuellen Verzeichnis gespeichert wird (`seeding_time.csv`). In der ersten Spalte der Datei stehen dabei die Dateinamen, in der zweiten die benötigten Sekunden. Dadurch dass die Zeit nicht schon direkt beim Programmstart gemessen und protokolliert wird, haben die Testpersonen die Gelegenheit, sich zunächst mit der Bedienung des Programms vertraut zu machen.

Beendet wird das Programm `Seeding-Tool` über den Menüpunkt `Datei` ▷ `beenden`. Dabei wird, sofern zuvor der Button `Bearbeitung starten` gedrückt worden ist, auch noch die Datei `seeding_time.csv` aktualisiert.

A.3 PLY-Dateiformat

Zum Speichern dreidimensionaler Modelle haben sich im Laufe der Zeit unterschiedliche Dateiformate etabliert. Das EvOpSeg-Tool sowie das Seeding-Tool nutzen eine Variante des PLY-Dateiformats[1], bei der die Daten in textueller Form (ASCII-Modus) gespeichert werden. Eine solche PLY-Datei beginnt mit einem Header, der die Struktur der gespeicherten Daten definiert. Im Anschluss daran folgen dann die Daten. Eine Beispiel-Datei für einen Würfel ist in Listing A.1 angegeben.

Jede PLY-Datei kann Kommentare beinhalten, die mit dem Schlüsselwort `comment` am Anfang einer jeden Kommentarzeile eingeleitet werden (Zeile 3 in Listing A.1). Im genannten Würfel-Beispiel wird in den Zeilen 4–7 definiert, in welcher Form die Eckpunkte des Dreiecksnetzes gespeichert sind: Jedes `vertex`-Element ist gegeben als ein Tripel aus reellwertigen Koordinaten im Raum (`property`-Einträge in den Zeilen 5–7). Die Zahl 8 in Zeile 4 gibt an, dass das Modell insgesamt 8 Eckpunkte beinhaltet. Die Koordinaten der Eckpunkte sind im Anschluss an den Header aufgeführt, wobei jede Zeile für einen einzelnen Punkt steht und der im Header angegebenen `vertex`-Element-Struktur entspricht.

Wie in den Zeilen 8 und 9 des Beispiel-Listings A.1 definiert, folgt auf die Koordinaten der Punkte die Angabe der zwölf[2] Dreiecksflächen. Dabei gibt der erste Parameter

[1] Weiterführende Informationen zum PLY-Dateiformat, das auch als *Polygon File Format* oder *Stanford Triangle Format* bekannt ist, sind beispielsweise unter [MB08] oder unter der URL `http://paulbourke.net/dataformats/ply/` zu finden.

[2] Da in dieser Arbeit Dreiecksnetze betrachtet werden, wird jede Würfelfläche aus zwei Dreiecken zusammengesetzt, so dass also insgesamt zwölf Dreiecke vorliegen.

```
1  ply
2  format ascii 1.0
3  comment Beispiel für eine PLY-Datei (hier: Würfel)
4  element vertex 8
5  property float x
6  property float y
7  property float z
8  element face 12
9  property list uchar int vertex_indices
10 end_header
11 1.0 -1.0 -1.0
12 -1.0 -1.0 -1.0
13 -1.0 1.0 -1.0
14 1.0 1.0 -1.0
15 1.0 -1.0 1.0
16 -1.0 -1.0 1.0
17 -1.0 1.0 1.0
18 1.0 1.0 1.0
19 3 0 1 3
20 3 1 2 3
21 3 4 0 7
22 3 0 3 7
23 3 5 4 6
24 3 4 7 6
25 3 1 5 2
26 3 5 6 2
27 3 3 2 7
28 3 2 6 7
29 3 4 5 0
30 3 5 1 0
```

Listing A.1: *PLY-Datei für einen Würfel.*

an, wie viele Eckpunkte eine Fläche hat (bei Dreiecksnetzen ist dieser Wert stets 3), und dahinter folgen die Indizes der `vertex`-Elemente, die die Eckpunkte der jeweiligen Dreiecksfläche darstellen.

Das PLY-Datenformat ist erweiterbar, so dass man neben den Elementen `vertex` und `face` auch weitere, problemspezifische Elemente definieren kann. Insbesondere können Modelle, die in einem erweiterten PLY-Format abgespeichert sind, auch weiterhin von gängigen Standard-Programmen geladen werden, da diese alle unbekannten Erweiterungen einfach ignorieren. Für das EvOpSeg-Tool ist insbesondere relevant, dass erhaltene Segmentierungen gespeichert werden können. Dies geschieht, indem die Nummern der Segmente gespeichert werden. Daneben sollen aber auch Informationen über die zugrunde liegenden Patches und Sub-Patches sowie Angaben zu Seed-Positionen und Gewichten in der PLY-Datei enthalten sein. Dazu erfolgt eine Erweiterung um das Element `additionalinformation`, die in Listing A.2 zu sehen ist. Es handelt sich um den Würfel aus dem vorangegangenen Listing, auf dem aber nun zwei Seeds angeordnet und entsprechend zwei Segmente berechnet worden sind. Ab Zeile 39 stehen dort Angaben zu der Segmentnummer, der Patch-Nummer und der Sub-Patch-Nummer eines jeden Dreiecks. Zusätzlich ist auch zu jedem Dreieck gespeichert, ob es einen Seed enthält und zu welchem Segment[3], welchem Patch und welchem Sub-Patch dieser Seed gehört;

[3]Die Segmentnummer des Seeds weicht hier von derjenigen des Dreiecks um 1 ab, da die Segmentnummer des Dreiecks im Gegensatz zu der des Seeds in erster Linie als Farbindex angesehen wird und

negative Werte geben an, dass es sich nicht um ein Seed-Dreieck handelt. Im genannten Beispiel lassen sich für Dreiecke, die Seeds enthalten, Informationen über die Zugehörigkeit der Seeds zu Segmenten, Patches und Sub-Patches auch über die entsprechenden Informationen der Dreiecke herleiten. Diese auf den ersten Blick redundant erscheinende Information wird allerdings in gewissen Situationen durchaus benötigt. Dies ist der Fall, wenn ein mit Seeds versehendes Modell gespeichert werden soll, ohne dass eine Berechnung der Segmentierung stattgefunden hat.

auch nicht-segmentierte Modelle gespeichert werden sollen, deren Dreiecke dann aber den Farbindex 0 erhalten.

```
1  ply
2  format ascii 1.0
3  comment Beispiel für eine erweiterte PLY–Datei (hier: Würfel)
4  element vertex 8
5  property float x
6  property float y
7  property float z
8  element face 12
9  property list uchar int vertex_indices
10 element additionalinformation
11 property uint component
12 property uint patch
13 property uint subpatch
14 property float seedweight
15 property uint componentnbrofseed
16 property uint patchnbrofseed
17 property uint subpatchnbrofseed
18 end_header
19 1.0 -1.0 -1.0
20 -1.0 -1.0 -1.0
21 -1.0 1.0 -1.0
22 1.0 1.0 -1.0
23 1.0 -1.0 1.0
24 -1.0 -1.0 1.0
25 -1.0 1.0 1.0
26 1.0 1.0 1.0
27 3 0 1 3
28 3 1 2 3
29 3 4 0 7
30 3 0 3 7
31 3 5 4 6
32 3 4 7 6
33 3 1 5 2
34 3 5 6 2
35 3 3 2 7
36 3 2 6 7
37 3 4 5 0
38 3 5 1 0
39 2 1 1 1.0 1 1 1
40 2 1 1 -9.0 -9 -9 -9
41 2 1 1 -9.0 -9 -9 -9
42 2 1 1 -9.0 -9 -9 -9
43 1 0 0 1.0 0 0 0
44 1 0 0 -9.0 -9 -9 -9
45 2 1 1 -9.0 -9 -9 -9
46 1 0 0 -9.0 -9 -9 -9
47 2 1 1 -9.0 -9 -9 -9
48 1 0 0 -9.0 -9 -9 -9
49 1 0 0 -9.0 -9 -9 -9
50 2 1 1 -9.0 -9 -9 -9
```

Listing A.2: *Erweiterte PLY-Datei für einen Würfel.*

Anhang B

Auswertungen

In Kapitel 8 sind einige Evaluationsergebnisse präsentiert worden. Eine deutlich detailliertere Übersicht über die entsprechenden Resultate des EvOpSeg-Verfahrens mit den in Kapitel 8 genannten Einstellungen wird in diesem Anhang gegeben. Zunächst geht Abschnitt B.1 auf die Fehlerwerte bei Verwendung des automatischen Seeding ein, bevor in den Abschnitten B.2 und B.3 die Fehlerwerte für das simulierte manuelle Seeding bzw. für ein manuelles Seeding durch Testpersonen präsentiert werden.

B.1 Evaluationsergebnisse (automatisches Seeding)

Die Evaluation des EvOpSeg-Verfahrens ist anhand des in [CGF09] vorgestellten Benchmark erfolgt. Zur Quantifizierung der Abweichung einer berechneten Segmentierung von Ground-Truth-Segmentierungen wurden dabei in dieser Dissertation die Metriken Rand-Index (RI), globaler Konsistenzfehler (GCE) und lokaler Konsistenzfehler (LCE) verwendet, die ebenfalls in [CGF09] beschrieben sind. In Tabelle B.1 auf der nächsten Seite sind jeweils die RI-, GCE- und LCE-Werte für das Standard-EvOpSeg-Verfahren, für das EvOpSeg-Verfahren mit gebundener Mutation (mit $\omega = 8$) sowie für das EvOpSeg-Verfahren mit zweistufiger Patch-Optimierung angegeben. In der unteren Zeile stehen die durchschnittlichen Werte für alle 380 Benchmark-Modelle. In den darüberliegenden Zeilen sind die pro Modellkategorie gemittelten Fehlerwerte angegeben. Wie in Kapitel 8 dargelegt, enthält der Benchmark insgesamt 4300 Ground-Truth-Segmentierungen, so dass durchschnittlich rund elf solcher Referenz-Segmentierungen pro Modell vorliegen.

Kategorie		Standard-EvOpSeg			gebundene Mutation			zweistufige Opt.		
		RI	GCE	LCE	RI	GCE	LCE	RI	GCE	LCE
1	Menschen	0.1311	0.2100	0.1427	0.1329	0.2181	0.1457	0.1449	0.2303	0.1551
2	Tassen	0.1552	0.0425	0.0251	0.1430	0.0359	0.0216	0.1448	0.0418	0.0242
3	Brillen	0.1612	0.1275	0.0515	0.1569	0.1178	0.0489	0.1744	0.1329	0.0579
4	Flugzeuge	0.1352	0.1723	0.0948	0.1266	0.1596	0.0875	0.1233	0.1533	0.0832
5	Ameisen	0.0573	0.0851	0.0354	0.0383	0.0639	0.0336	0.0340	0.0620	0.0355
6	Stühle	0.1301	0.1144	0.0572	0.0972	0.0906	0.0484	0.0966	0.0893	0.0464
7	Kraken	0.0531	0.0538	0.0330	0.0312	0.0423	0.0250	0.0326	0.0439	0.0252
8	Tische	0.2198	0.0465	0.0330	0.1801	0.0430	0.0290	0.1344	0.0291	0.0181
9	Teddys	0.1015	0.1087	0.0412	0.0637	0.0742	0.0343	0.0873	0.0935	0.0374
10	Hände	0.1311	0.1660	0.0999	0.1248	0.1584	0.1002	0.1237	0.1495	0.0981
11	Zangen	0.1569	0.1932	0.1197	0.1886	0.1790	0.1050	0.1438	0.1858	0.1158
12	Fische	0.3415	0.1820	0.1146	0.3037	0.1774	0.1102	0.2975	0.1701	0.1031
13	Vögel	0.1321	0.1753	0.0983	0.1112	0.1530	0.0891	0.1214	0.1650	0.0913
15	Armadillo	0.1064	0.2774	0.1673	0.1065	0.2698	0.1634	0.1176	0.2853	0.1795
16	Büsten	0.3126	0.2075	0.1431	0.2932	0.2000	0.1286	0.2661	0.1957	0.1220
17	Mechanik	0.2077	0.0622	0.0450	0.1683	0.0543	0.0385	0.1379	0.0364	0.0231
18	Bolzen	0.2687	0.1085	0.0596	0.2509	0.0978	0.0523	0.1784	0.0611	0.0290
19	Vasen	0.2358	0.1299	0.0817	0.2170	0.1205	0.0725	0.2034	0.1003	0.0585
20	Vierbeiner	0.1756	0.2574	0.1762	0.1777	0.2575	0.1672	0.1886	0.2701	0.1785
Durchschnitt		0.1691	0.1432	0.0852	0.1533	0.1323	0.0790	0.1448	0.1313	0.0780

Tabelle B.1: *Vergleich der Evaluationsergebnisse für die 19 Kategorien. Hier liegt jeweils das automatische Seeding zugrunde.*

B.2 Evaluationsergebnisse (simuliertes manuelles Seeding)

In Tabelle B.2 sind die RI-, GCE- und LCE-Werte für das Standard-EvOpSeg-Verfahren, für das EvOpSeg-Verfahren mit gebundener Mutation sowie für das EvOpSeg-Verfahren mit zweistufiger Patch-Optimierung angegeben, wobei hier das simulierte manuelle Seeding zum Einsatz gekommen ist. Analog zum vorangegangenen Abschnitt B.1 handelt es sich bei den Zahlen in der unteren Zeile um die durchschnittlichen Werte für alle 380 Benchmark-Modelle, während in den darüberliegenden Zeilen die pro Modellkategorie gemittelten Fehlerwerte angegeben sind.

Kategorie		Standard-EvOpSeg			gebundene Mutation			zweistufige Opt.		
		RI	GCE	LCE	RI	GCE	LCE	RI	GCE	LCE
1	Menschen	0.1463	0.2048	0.1401	0.1546	0.2130	0.1452	0.1647	0.2159	0.1483
2	Tassen	0.1530	0.0346	0.0203	0.1195	0.0248	0.0150	0.1560	0.0214	0.0125
3	Brillen	0.0993	0.0648	0.0445	0.1131	0.0618	0.0404	0.1133	0.0643	0.0435
4	Flugzeuge	0.0904	0.1029	0.0660	0.0956	0.1005	0.0668	0.0947	0.1024	0.0694
5	Ameisen	0.0203	0.0327	0.0233	0.0224	0.0384	0.0265	0.0244	0.0408	0.0282
6	Stühle	0.0825	0.0697	0.0408	0.0836	0.0680	0.0378	0.0861	0.0674	0.0385
7	Kraken	0.0240	0.0335	0.0222	0.0240	0.0336	0.0219	0.0319	0.0340	0.0228
8	Tische	0.1008	0.0226	0.0168	0.0803	0.0212	0.0150	0.0884	0.0186	0.0127
9	Teddys	0.0406	0.0489	0.0329	0.0438	0.0490	0.0333	0.0621	0.0641	0.0416
10	Hände	0.1109	0.1277	0.0879	0.1166	0.1394	0.0995	0.1453	0.1314	0.0889
11	Zangen	0.0936	0.1201	0.0841	0.1011	0.1148	0.0830	0.1238	0.1181	0.0821
12	Fische	0.1791	0.1171	0.0749	0.1889	0.1021	0.0680	0.1991	0.1201	0.0771
13	Vögel	0.0809	0.1310	0.0880	0.0734	0.1154	0.0773	0.1017	0.1355	0.0867
15	Armadillo	0.1092	0.2327	0.1646	0.1038	0.2270	0.1557	0.1259	0.2473	0.1707
16	Büsten	0.2260	0.1442	0.0986	0.2549	0.1345	0.0912	0.2697	0.1327	0.0867
17	Mechanik	0.0916	0.0264	0.0171	0.0953	0.0229	0.0135	0.1002	0.0250	0.0143
18	Bolzen	0.0947	0.0457	0.0322	0.1079	0.0400	0.0214	0.1018	0.0496	0.0305
19	Vasen	0.1164	0.0654	0.0430	0.1261	0.0607	0.0412	0.1424	0.0582	0.0394
20	Vierbeiner	0.1504	0.2011	0.1366	0.1785	0.1994	0.1327	0.1745	0.2254	0.1477
Durchschnitt		0.1057	0.0961	0.0649	0.1096	0.0929	0.0623	0.1213	0.0985	0.0653

Tabelle B.2: *Vergleich der Evaluationsergebnisse für die 19 Kategorien. Hier liegt jeweils das simulierte manuelle Seeding zugrunde.*

B.3 Evaluationsergebnisse (manuelles Seeding)

Für das in Abschnitt 8.5 beschriebene manuelle Seeding durch Testpersonen ist zusätzlich zur Auswertung nach den Modellkategorien auch eine Auswertung nach den Testpersonen erfolgt. Diese Ergebnisse sowie die Abweichungen der manuell platzierten Seeds von den simuliert-manuell gesetzten Seeds sind im Folgenden angegeben.

B.3.1 Auswertung nach Kategorien

In Tabelle B.3 sind die RI-, GCE- und LCE-Werte für das Standard-EvOpSeg-Verfahren, für das EvOpSeg-Verfahren mit gebundener Mutation sowie für das EvOpSeg-Verfahren mit zweistufiger Patch-Optimierung angegeben, wobei hier ein manuelles Seeding durch Testpersonen zum Einsatz gekommen ist.

Kategorie		Standard-EvOpSeg			gebundene Mutation			zweistufige Opt.		
		RI	GCE	LCE	RI	GCE	LCE	RI	GCE	LCE
1	Menschen	0.1128	0.1886	0.1304	0.1491	0.2008	0.1313	0.1577	0.1955	0.1287
2	Tassen	0.0967	0.0185	0.0106	0.1170	0.0187	0.0108	0.1196	0.0154	0.0085
3	Brillen	0.0603	0.0441	0.0320	0.1315	0.0545	0.0371	0.1394	0.0647	0.0450
4	Flugzeuge	0.0572	0.0835	0.0567	0.1153	0.0847	0.0566	0.1621	0.1055	0.0696
5	Ameisen	0.0879	0.0336	0.0236	0.0264	0.0336	0.0236	0.0261	0.0337	0.0240
6	Stühle	0.1659	0.0599	0.0343	0.0784	0.0600	0.0345	0.0794	0.0581	0.0345
7	Kraken	0.0865	0.0321	0.0207	0.0216	0.0320	0.0215	0.0210	0.0314	0.0206
8	Tische	0.1601	0.0149	0.0097	0.0713	0.0148	0.0096	0.0680	0.0146	0.0093
9	Teddys	0.0763	0.0485	0.0320	0.0517	0.0481	0.0324	0.0509	0.0495	0.0336
10	Hände	0.1316	0.1124	0.0784	0.1388	0.1197	0.0870	0.1403	0.1199	0.0860
11	Zangen	0.1297	0.0875	0.0704	0.1978	0.0869	0.0680	0.2205	0.0898	0.0653
12	Fische	0.0929	0.0910	0.0616	0.2054	0.0863	0.0571	0.2268	0.0895	0.0586
13	Vögel	0.1016	0.1089	0.0766	0.0934	0.1073	0.0742	0.1083	0.1128	0.0752
15	Armadillo	0.0971	0.2252	0.1558	0.1113	0.2299	0.1609	0.1230	0.2464	0.1711
16	Büsten	0.1408	0.1583	0.1037	0.2863	0.1477	0.0979	0.3075	0.1290	0.0833
17	Mechanik	0.1590	0.0204	0.0122	0.0961	0.0221	0.0133	0.1004	0.0216	0.0123
18	Bolzen	0.1892	0.0283	0.0190	0.1067	0.0176	0.0099	0.1207	0.0308	0.0164
19	Vasen	0.0909	0.0550	0.0383	0.1439	0.0622	0.0405	0.1802	0.0618	0.0370
20	Vierbeiner	0.1053	0.2006	0.1374	0.1761	0.1951	0.1348	0.2046	0.2028	0.1401
Durchschnitt		0.1127	0.0848	0.0581	0.1220	0.0854	0.0580	0.1345	0.0880	0.0589

Tabelle B.3: *Vergleich der Evaluationsergebnisse für die 19 Kategorien. Zugrunde liegt jeweils ein echtes manuelles Seeding.*

B.3.2 Auswertung nach Testpersonen (Standard-EvOpSeg-Verfahren)

Während Tabelle B.3 die Ergebnisse nach den Modellkategorien gruppiert auflistet, sind die Ergebnisse des Standard-EvOpSeg-Verfahrens in Tabelle B.4 nach Modellgruppen aufgelistet. Jede Modellgruppe enthält genau ein Modell je Kategorie und ist von einer Testperson bearbeitet worden. Die von den einzelnen Testpersonen für das Platzieren der Seeds benötigte durchschnittliche Zeit ist ebenfalls angegeben – eine durchschnittliche Testperson benötigt demnach rund 100 Sekunden pro Modell.

B.3.3 Vergleich mit simuliertem manuellem Seeding

Im Rahmen der Dissertation sind die von Testpersonen platzierten Seeds mit denen des simulierten manuellen Seeding anhand des Hausdorff-Average-Abstandes sowie des modifizierten Hausdorff-Average-Abstandes verglichen worden. Tabelle B.5 gibt die über die einzelnen Modellgruppen gemittelten Werte an.

Modell-gruppe	RI	GCE	LCE	Seeding-Dauer (Sek./Modell)
1	0.0883	0.0783	0.0570	45.89
2	0.0879	0.0917	0.0620	49.89
3	0.1159	0.0842	0.0574	151.79
4	0.1073	0.1095	0.0675	93.32
5	0.1091	0.0905	0.0625	83.32
6	0.0927	0.0874	0.0628	64.26
7	0.1550	0.0711	0.0480	84.68
8	0.0884	0.0569	0.0402	107.74
9	0.1099	0.0820	0.0587	102.37
10	0.0696	0.0899	0.0649	272.83
11	0.1477	0.0727	0.0502	48.72
12	0.0964	0.0609	0.0541	160.61
13	0.0892	0.0847	0.0574	83.68
14	0.0993	0.0948	0.0617	82.42
15	0.1227	0.0965	0.0651	85.00
16	0.1327	0.0927	0.0622	88.95
17	0.1609	0.0968	0.0627	78.56
18	0.1755	0.0637	0.0433	70.24
19	0.0888	0.0862	0.0595	172.26
20	0.1032	0.0905	0.0640	83.21

Tabelle B.4: *Seeding-Dauer und Segmentierungsqualität für das Standard-EvOpSeg-Verfahren bei einem manuellen Seeding durch Testpersonen.*

Modell-gruppe	Hausdorff-Avg.		mod. Hausdorff-Avg.	
	auto.	sim. man.	auto.	sim. man.
1	0.2705	0.1477	0.0990	0.0239
2	0.2242	0.1598	0.0797	0.0473
3	0.2989	0.1517	0.0932	0.0275
4	0.2676	0.1735	0.1010	0.0307
5	0.3573	0.2683	0.1104	0.0698
6	0.3185	0.1621	0.0993	0.0325
7	0.3038	0.1760	0.1001	0.0310
8	0.3584	0.2247	0.1245	0.0444
9	0.3255	0.2226	0.0858	0.0500
10	0.2650	0.1525	0.1020	0.0387
11	0.3716	0.2878	0.1043	0.0527
12	0.2956	0.1608	0.1022	0.0292
13	0.2769	0.1935	0.0894	0.0445
14	0.2530	0.1613	0.1015	0.0389
15	0.2616	0.1632	0.1048	0.0365
16	0.3108	0.1807	0.1025	0.0549
17	0.3132	0.2092	0.0895	0.0427
18	0.3285	0.2430	0.0805	0.0629
19	0.3000	0.1848	0.1020	0.0294
20	0.2677	0.1606	0.0968	0.0375
Durchschn.	0.2984	0.1892	0.0984	0.0413

Tabelle B.5: *Auswertung des Hausdorff-Average-Abstandes sowie des modifizierten Hausdorff-Average-Abstandes für das automatische sowie das simulierte manuelle Seeding bezogen auf manuell gesetzte Seeds.*

Literaturverzeichnis

[AE84] Franz Aurenhammer und Herbert Edelsbrunner: *An Optimal Algorithm for Constructing the Weighted Voronoi Diagram in the Plane.* Pattern Recognition, 17(2):251–257, 1984.

[AFS06] Marco Attene, Bianca Falcidieno und Michela Spagnuolo: *Hierarchical mesh segmentation based on fitting primitives.* Vis. Comput., 22(3):181–193, 2006.

[AKM+06] Marco Attene, Sagi Katz, Michaela Mortara, Giuseppe Patanè, Michela Spagnuolo und Ayellet Tal: *Mesh Segmentation - A Comparative Study.* In: *SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*, Seite 7, Washington, DC, USA, 2006. IEEE Computer Society.

[AM91] Stephan Abramowski und Heinrich Müller: *Geometrisches Modellieren,* Reihe Informatik, Band 75. BI-Wissenschaftsverlag, Mannheim, 1991.

[Amd67] Gene M. Amdahl: *Validity of the single processor approach to achieving large scale computing capabilities.* In: *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), Seiten 483–485, New York, NY, USA, 1967. ACM.

[AMH02] Thomas Akenine-Möller und Eric Haines: *Real-Time Rendering, Second Edition.* A K Peters, Natick, Massachusetts, 2002.

[APP+07] Er Agathos, Ioannis Pratikakis, Stavros Perantonis, Nikolaos Sapidis und Philip Azariadis: *3D Mesh Segmentation Methodologies for CAD applications.* In: *Computer-Aided Design & Applications 4, 6*, Seiten 827–841, 2007.

[BAT11] Filippo Bergamasco, Andrea Albarelli und Andrea Torsello: *Semi-supervised Segmentation of 3D Surfaces using a Weighted Graph Representation.* In: *Proceedings of the 8th international conference on Graph-based representations in pattern recognition*, GbRPR'11, Seiten 225–234, Berlin, Heidelberg, 2011. Springer-Verlag.

[Bau75] Bruce G. Baumgart: *Winged-Edge Polyhedron Representation for Computer Vision.* In: *National Computer Conference*, Mai 1975.

[Bäc96] Thomas Bäck: *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, Oxford, UK, 1996.

[BEB06] Ron Breukelaer, Michael Emmerich und Thomas Bäck: *On Interactive Evolution Strategies.* In: *EvoWorkshops*, Seiten 530–541, 2006.

[BHN+99] Thomas Bäck, Werner Haase, Boris Naujoks, Luca Onesti und Alessio Turchet: *Evolutionary Algorithms Applied to Academic and Industrial Test Cases.* In: K. Miettinen, M. M. Mäkelä, P. Neittaanmäki und J. Périaux (Herausgeber): *Evolutionary Algorithms in Engineering and Computer Science*, Kapitel 17, Seiten 383–397. Wiley, Chichester, 1999.

[BPR+06] Mario Botsch, Mark Pauly, Christian Rossl, Stephan Bischoff und Leif Kobbelt: *Geometric Modeling Based on Triangle Meshes.* In: *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[BRS93] Thomas Bäck, Günter Rudolph und Hans-Paul Schwefel: *Evolutionary Programming and Evolution Strategies: Similarities and Differences.* In: *Proceedings of the Second Annual Conference on Evolutionary Programming*, 1993.

[BS93] Thomas Bäck und Hans-Paul Schwefel: *An Overview of Evolutionary Algorithms for Parameter Optimization.* Evol. Comput., 1(1):1–23, 1993.

[BSMM99] Ilja N. Bronstein, Konstantin A. Semendjajew, Gerhard Musiol und Heiner Mühlig: *Taschenbuch der Mathematik.* Verlag Harri Deutsch, Frankfurt am Main, Thun, 4. Auflage, 1999.

[BVLD09] Halim Benhabiles, Jean-Philippe Vandeborre, Guillaume Lavoué und Mohamed Daoudi: *A framework for the objective evaluation of segmentation algorithms using a ground-truth of human segmented 3D-models.* In: *IEEE International Conference on Shape Modeling and Applications (Shape Modeling International 2009)*, Beijing, China, June 26-28 2009. short paper.

[BVLD10] Halim Benhabiles, Jean-Philippe Vandeborre, Guillaume Lavoué und Mohamed Daoudi: *A comparative study of existing metrics for 3D-mesh segmentation evaluation.* The Visual Computer - International Journal of Computer Graphics, 2010.

[CGF09] Xiaobai Chen, Aleksey Golovinskiy und Thomas Funkhouser: *A Benchmark for 3D Mesh Segmentation.* ACM Transactions on Graphics (Proc. SIGGRAPH), 28(3), August 2009.

[CLJ07] Zhi-Quan Cheng, Hua-Feng Liu und Shi-Yao Jin: *The progressive mesh compression based on meaningful segmentation.* Vis. Comput., 23(9):651–660, 2007.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest und Clifford Stein: *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2. Auflage, 2001.

[Cox61] Harold Scott MacDonald Coxeter: *Non-Euclidean Geometry*. Mathematical Expositions, No. 2. Toronto; printed in Great Britain, 4. Auflage, 1961.

[CP97] Erick Cantú-Paz: *A Survey of Parallel Genetic Algorithms*, 1997.

[CS92] Xin Chen und Francis Schmitt: *Intrinsic Surface Properties from Surface Triangulation*. In: G. Sandini (Herausgeber): *Computer Vision – ECCV92*, Band 588 der Reihe *Lecture Notes in Computer Science*, Seiten 739–743. Springer Berlin / Heidelberg, 1992. 10.1007/3-540-55426-283.

[Deb02] Kalyanmoy Deb: *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Ltd, 2002.

[Die06] Reinhard Diestel: *Graphentheorie (3. Aufl.)*. Springer, 2006.

[Dij59] Edsger W. Dijkstra: *A Note on Two Problems in Connexion with Graphs*. Numerische Mathematik, 1(1):269–271, 1959.

[DKT05] Mathieu Desbrun, Eva Kanso und Yiying Tong: *Discrete Differential Forms for Computational Modeling*. In: *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[DLJD00] Dumitru Dumitrescu, Beatrice Lazzerini, Lakhmi C. Jain und Anca Dumitrescu: *Evolutionary Computation*. CRC Press, 2000.

[DW05] Chen-shi Dong und Guo-zhao Wang: *Curvatures estimation on triangular mesh*. Journal of Zhejiang University - Science A, 6:128–136, 2005. 10.1007/BF02887228.

[EGSG00] Michael Emmerich, Monika Grotzner, Martin Schütz und Bernd Groß: *Mixed-Integer Evolution Strategy for Chemical Plant Optimization with Simulators*. In: Ian Parmee (Herausgeber): *Evolutionary Design and Manufacture (ACDM)*, 2000.

[EM12] Kai Engel und Heinrich Müller: *Evolutionary 3D-Shape Segmentation Using Satellite Seeds*. In: Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia und Mario Pavone (Herausgeber): *Parallel Problem Solving from Nature - PPSN XII*, Band 7492 der Reihe *Lecture Notes in Computer Science*, Seiten 438–447. Springer, 2012.

[FKS+04] Thomas A. Funkhouser, Michael M. Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz und David P. Dobkin: *Modeling by Example*. ACM Trans. Graph., 23(3):652–663, 2004.

[FvDFH95] James Foley, Andries van Dam, Steven Feiner und John Hughes: *Computer Graphics: Principles and Practice, second edition in C.* Addison-Wesley Professional, 1995.

[GB06] Juan Francisco Garamendi und Jose Luis Bosque: *Parallel Implementation of Evolutionary Strategies on Heterogeneous Clusters with Load Balancing.* In: *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, IPDPS'06, Seiten 242–242, Washington, DC, USA, 2006. IEEE Computer Society.

[GBP07] Daniela Giorgi, Silvia Biasotti und Laura Paraboschi: *SHape REtrieval Contest 2007: Watertight Models Track.* In: R. C. Veltkamp und F. B. T. Haar (Herausgeber): *SHREC2007: 3D shape retrieval contest*, Technical Report UU-CS-2007-015, Seiten 5–10, 2007.

[GF08] Aleksey Golovinskiy und Thomas Funkhouser: *Randomized Cuts for 3D Mesh Analysis.* ACM Trans. Graph., 27(5):1–12, 2008.

[GGGZ05] Timothy Gatzke, Cindy Grimm, Michael Garland und Steve Zelinka: *Curvature Maps for Local Shape Comparison.* In: *In Shape Modeling International*, Seiten 244–256, 2005.

[GKK04] Ingrid Gerdes, Frank Klawonn und Rudolf Kruse: *Evolutionäre Algorithmen.* Computational Intelligence. Vieweg+Teubner, 1. Auflage, July 2004.

[GS90] Martina Gorges-Schleuter: *Genetic Algorithms and Population Structures – A Massively Parallel Algorithm.* Doktorarbeit, Universität Dortmund, 1990.

[Gum00] Stefan Gumhold: *Mesh Compression.* Doktorarbeit, Eberhard-Karls-Universität zu Tübingen, 2000.

[Heb09] Matthias Hebbel: *Evolutionäre Algorithmen zur Optimierung von Modellen für laufende Roboter.* Doktorarbeit, Technische Universität Dortmund, 2009.

[Hei07] Rüdiger Heintz: *Neues Verfahren zur invarianten Objekterkennung und -lokalisierung auf der Basis lokaler Merkmale.* Doktorarbeit, Universität Karlsruhe (TH), 2007.

[HKKR93] Daniel P. Huttenlocher, Gregory A. Klanderman, Gregory A. Kl und William J. Rucklidge: *Comparing Images Using the Hausdorff Distance.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 15:850–863, 1993.

[HL09] Simon Hoffmann und Rainer Lienhard: *OpenMP.* Informatik im Fokus. Springer-Verlag Berlin Heidelberg, 2009.

[Hoc06] Ronald Hochreiter: *Audible Convergence for Optimal Base Melody Extension with Statistical Genre-specific Interval Distance Evaluation.* In: *Proceedings of the 2006 International Conference on Applications of Evolutionary Computing*, EuroGP'06, Seiten 712–716, Berlin, Heidelberg, 2006. Springer-Verlag.

[Hol75] John H. Holland: *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, USA, 1975.

[Hor69] C. B. Horan: *Multidimensional Scaling: Combining Observations When Individuals Have Different Perceptual Structures.* Psychometrika, 34:139–165, 1969. 10.1007/BF02289341.

[HR84] Donald D. Hoffman und Whitman A. Richards: *Parts Of Recognition.* Cognition, 18:65–96, 1984.

[KGV83] Scott Kirkpatrick, C. Daniel Gelatt und Mario P. Vecchi: *Optimization by Simulated Annealing.* Science, 220:671–680, 1983.

[KHS10] Evangelos Kalogerakis, Aaron Hertzmann und Karan Singh: *Learning 3D Mesh Segmentation and Labeling.* ACM Transactions on Graphics, 29(3), 2010.

[KJS07] Vladislav Kreavoy, Dan Julius und Alla Sheffer: *Model Composition from Interchangeable Components.* Computer Graphics and Applications, Pacific Conference on, 0:129–138, 2007.

[KLT05] Sagi Katz, George Leifman und Ayellet Tal: *Mesh segmentation using feature point and core extraction.* The Visual Computer, 21(8-10):649–658, 2005.

[KSNS07] Evangelos Kalogerakis, Patricio Simari, Derek Nowrouzezahrai und Karan Singh: *Robust Statistical Estimation of Curvature on Discretized Surfaces.* In: *Proceedings of the fifth Eurographics symposium on Geometry processing*, Seiten 13–22, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[KT03] Sagi Katz und Ayellet Tal: *Hierarchical Mesh Decomposition Using Fuzzy Clustering and Cuts.* ACM Transactions on Graphics (Proc. SIGGRAPH), 22(3):954–961, Juli 2003.

[KT09] Lotan Kaplansky und Ayellet Tal: *Mesh Segmentation Refinement.* Comput. Graph. Forum, 28(7):1995–2003, 2009.

[LDB03] Guillaume Lavoué, Florent Dupont und Atilla Baskurt: *Constant Curvature Region Decomposition of 3D-Meshes by a Mixed Approach Vertex-Triangle.* Technischer Bericht RR-LIRIS-2003-001, LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Université Lumière Lyon 2/Ecole Centrale de Lyon, November 2003. Validé par le comité de lecture de WSCG 2004.

[LEB+06] Rui Li, Michael T. M. Emmerich, Ernst G. P. Bovenkamp, Jeroen Eggermont, Thomas Bäck, Jouke Dijkstra und Johan H. C. Reiber: *Mixed-Integer Evolution Strategies and Their Application to Intravascular Ultrasound Image Analysis*. In: *Proceedings of the 2006 international conference on Applications of Evolutionary Computing*, EuroGP'06, Seiten 415–426, Berlin, Heidelberg, 2006. Springer-Verlag.

[LEEB06] Rui Li, Michael T.M. Emmerich, Jeroen Eggermont und Ernst G.P. Bovenkamp: *Mixed-Integer Optimization of Coronary Vessel Image Analysis using Evolution Strategies*. In: *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, Seiten 1645–1652, New York, NY, USA, 2006. ACM.

[LHMR08] Yu-Kun Lai, Shi-Min Hu, Ralph R. Martin und Paul L. Rosin: *Fast Mesh Segmentation Using Random Walks*. In: *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, Seiten 183–191, New York, NY, USA, 2008. ACM.

[Liu09] Rong Liu: *Spectral Mesh Segmentation*. Doktorarbeit, Simon Fraser University, Burnaby, Canada, 2009.

[LOJ+07] Dudy Lim, Yew-Soon Ong, Yaochu Jin, Bernhard Sendhoff und Bu-Sung Lee: *Efficient Hierarchical Parallel Genetic Algorithms using Grid Computing*. Future Generation Computer Systems, 23(4):658–670, may 2007.

[LPRM02] Bruno Lévy, Sylvain Petitjean, Nicolas Ray und Jérome Maillot: *Least Squares Conformal Maps for Automatic Texture Atlas Generation*. In: *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, Seiten 362–371, New York, NY, USA, 2002. ACM.

[LV98] Francis Lazarus und Anne Verroust: *Three-Dimensional Metamorphosis: a Survey*. The Visual Computer, 14(8/9):373–389, 1998.

[LWTH01] Xuetao Li, Tong Wing Woon, Tiow Seng Tan und Zhiyong Huang: *Decomposing Polygon Meshes for Interactive Applications*. In: *SI3D*, Seiten 35–42, 2001.

[LZHM06] Yu-Kun Lai, Qian-Yi Zhou, Shi-Min Hu und Ralph R. Martin: *Feature Sensitive Mesh Segmentation*. In: *SPM '06: Proceedings of the 2006 ACM symposium on Solid and physical modeling*, Seiten 17–25, New York, NY, USA, 2006. ACM.

[LZSCO09] Rong Liu, Hao Zhang, Ariel Shamir und Daniel Cohen-Or: *A Part-Aware Surface Metric for Shape Analysis*. Computer Graphics Forum (Special Issue of Eurographics 2009), 28(2):397–406, 2009.

[Män88] Martti Mäntylä: *An Introduction to Solid Modeling*. Principles of Computer Science Series. Computer Science Press, 1988.

[MB08] Kenton McHenry und Peter Bajcsy: *An Overview of 3D Data Content, File Formats and Viewers.* Technischer Bericht isda08-002, Image Spatial Data Analysis Group, National Center for Supercomputing Applications, 1205 W Clark, Urbana, IL 61801, 2008.

[MFTM01] David Martin, Charless Fowlkes, Doron Tal und Jitendra Malik: *A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics.* In: *in Proc. 8th Intl Conf. Computer Vision*, Seiten 416–423, 2001.

[MPS+04] Michela Mortara, Giuseppe Patanè, Michela Spagnuolo, Bianca Falcidieno und Jarek Roman Rossignac: *Plumber: a method for a multi-scale decomposition of 3D shapes into tubular primitives and bodies.* In: *SM '04: Proceedings of the ninth ACM symposium on Solid modeling and applications*, Seiten 339–344, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[Nis94] Volker Nissen: *Evolutionäre Algorithmen - Darstellung, Beispiele, betriebswirtschaftliche Anwendungsmöglichkeiten.* Deutscher Universitäts-Verlag, Wiesbaden, 1994.

[NORS11] Frank Neumann, Pietro Simone Oliveto, Günter Rudolph und Dirk Sudholt: *On the Effectiveness of Crossover for Migration in Parallel Evolutionary Algorithms.* In: *GECCO*, Seiten 1587–1594, 2011.

[Ort02] Matthias Ortmann: *Die Anwendung von Evolutionstrategien zur adaptiven, echtzeitfähigen Trajektorienplanung von Manipulatorarmsystemen.* Doktorarbeit, Ruhr-Universität Bochum, 2002.

[PADC05] Manuel Parrilla, Joaquín Aranda und Sebastian Dormido-Canto: *Parallel Evolutionary Computation: Application of an EA to Controller Design.* In: José Mira und José R. Álvarez (Herausgeber): *IWINAC (2)*, Band 3562 der Reihe *Lecture Notes in Computer Science*, Seiten 153–162. Springer, 2005.

[Pri04] Christian Prins: *A Simple and Effective Evolutionary Algorithm for the Vehicle Routing Problem.* Computers & Operations Research, 31(12):1985–2002, Oktober 2004.

[Ran71] William M. Rand: *Objective Criteria for the Evaluation of Clustering Methods.* Journal of the American Statistical Association, 66:622–626, 1971.

[Rec73] Ingo Rechenberg: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart, 1973.

[RHQ+05] Chris Rupp, Jürgen Hahn, Stefan Queins, Mario Jeckle und Barbara Zengler: *UML 2 glasklar.* Hanser Fachbuchverlag, 2. Auflage, Juni 2005.

[Rud91] Günter Rudolph: *Global Optimization by means of Distributed Evolution Strategies.* In: Hans-Paul Schwefel und Reinhard Männer (Herausgeber): *Proceedings of the 1st PPSN Conference*, Lecture Notes in Computer Science, Seiten 209–213. Springer, 1991.

[Rus04] Szymon Rusinkiewicz: *Estimating Curvatures and Their Derivatives on Triangle Meshes.* In: *Symposium on 3D Data Processing, Visualization, and Transmission*, September 2004.

[Sch75] Hans-Paul Schwefel: *Evolutionsstrategie und numerische Optimierung.* Dr.-Ing. Dissertation, Technische Universität Berlin, Fachbereich Verfahrenstechnik, 1975.

[Sch81] Hans-Paul Schwefel: *Numerical Optimization of Computer Models.* John Wiley & Sons, Inc., New York, NY, USA, 1981.

[Sch95] Hans-Paul Schwefel: *Evolution and Optimum Seeking.* Sixth-Generation Computer Technology. Wiley Interscience, New York, 1995.

[Sch96] Martin Schütz: *Eine Evolutionsstrategie für gemischt-ganzzahlige Optimierungsprobleme mit variabler Dimension.* Technischer Bericht, Fachbereich Informatik, Universität Dortmund, 1996.

[SELC12] Oliver Schütze, Xavier Esquivel, Adriana Lara und Carlos A. Coello Coello: *Using the Averaged Hausdorff Distance as a Performance Measure in Evolutionary Multiobjective Optimization.* IEEE Trans. Evolutionary Computation, 16(4):504–522, 2012.

[SGG00] Abraham Silberschatz, Peter B. Galvin und Greg Gagne: *Applied Operating System Concepts.* Wiley, New York [u.a.], 1. Auflage, 2000.

[Sha07] Ariel Shamir: *Strategies for Mesh Segmentation.* International Summer School on Shape Modeling and Reasoning, 2007. http://shapesummerschool07.disi.unige.it/slides/shamir/StrategiesShamir.pdf (Stand: 22.04.2014).

[Sha08] Ariel Shamir: *A survey on Mesh Segmentation Techniques.* Comput. Graph. Forum, 27(6):1539–1556, 2008.

[SHF94] Eberhard Schöneburg, Frank Heinzmann und Sven Feddersen: *Genetische Algorithmen und Evolutionsstrategien - Eine Einführung in Theorie und Praxis der simulierten Evolution.* Addison-Wesley, 1994.

[SN06] Márta Szilvási-Nagy: *About Curvatures on Triangle Meshes.* KoG Scientific-Professional Journal of Croatian Society for Geometry and Graphics, 10:13–18, 2006.

[SNKS09] Patricio D. Simari, Derek Nowrouzezahrai, Evangelos Kalogerakis und Karan Singh: *Multi-objective shape segmentation and labeling.* Comput. Graph. Forum, 28(5):1415–1425, 2009.

[Spr99] Joachim Sprave: *Ein einheitliches Modell für Populationsstrukturen in Evolutionären Algorithmen*. Doktorarbeit, Universität Dortmund, 1999.

[SPW10] Xiaopeng Sun, J. Pan und Xiaopeng Wei: *3D Mesh Skeleton Extraction Using Prominent Segmentation*. Comput. Sci. Inf. Syst., 7(1):63–74, 2010.

[SS08] Patricio Simari und Karan Singh: *Multi-objective shape segmentation*. Technischer Bericht, Departement of Computer Science, University of Toronto, 2008.

[SSCO08] Lior Shapira, Ariel Shamir und Daniel Cohen-Or: *Consistent Mesh Partitioning and Skeletonisation using the Shape Diameter Function*. The Visual Computer, 24(4):249–259, 2008.

[SSGH01] Pedro V. Sander, John Snyder, Steven J. Gortler und Hugues Hoppe: *Texture Mapping Progressive Meshes*. In: *SIGGRAPH*, Seiten 409–416, 2001.

[STK02] Shymon Shlafman, Ayellet Tal und Sagi Katz: *Metamorphosis of Polyhedral Surfaces using Decomposition*. In: *Computer Graphics Forum*, Seiten 219–228, 2002.

[Tak98] Hideyuki Takagi: *Interactive Evolutionary Computation: System optimization Based on Human Subjective Evaluation*. In: *IEEE Int'l Conf. On Intelligent Eng. Sys (INES '98)*, 1998.

[TCG08] J. Thériault, Farida Cheriet und François Guibault: *Automatic Detection of the Back Valley on Scoliotic Trunk Using Polygonal Surface Curvature*. In: Aurélio Campilho und Mohamed Kamel (Herausgeber): *Image Analysis and Recognition*, Band 5112 der Reihe *Lecture Notes in Computer Science*, Seiten 779–788. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-69812-8_77.

[TLK09] Min Tang, Minkyoung Lee und Young J. Kim: *Interactive Hausdorff Distance Computation for General Polygonal Models*. In: *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, Seiten 74:1–74:9, New York, NY, USA, 2009. ACM.

[Urs03] Rasmus K. Ursem: *Models for Evolutionary Algorithms and Their Applications in System Identification and Control Optimization*. Doktorarbeit, University of Aarhus, Dänemark, 2003.

[VBGS08] Romain Vergne, Pascal Barla, Xavier Granier und Christophe Schlick: *Apparent relief: a shape descriptor for stylized shading*. In: *Proceedings of the 6th international symposium on Non-photorealistic animation and rendering*, NPAR '08, Seiten 23–29, New York, NY, USA, 2008. ACM.

[Wat02] Alan Watt: *3D-Computergrafik*. Addison-Wesley, 3 Auflage, 2002.

[WMR01] Klaus Weinert, Jörn Mehnen und Günter Rudolph: *Dynamic Neighborhood Structures in Parallel Evolution Strategies.* Complex Systems, 13(3):227–243, 2001.

[WP99] Alan Watt und Fabio Policarpo: *The Computer Image.* Addison-Wesley, 1999.

[WPP+07] Huai-Yu Wu, Chunhong Pan, Jia Pan, Qing Yang und Songde Ma: *A sketch-based interactive framework for real-time mesh segmentation.* In: *Computer Graphics International*, 2007.

[YSOM10] Caiyun Yang, Hiromasa Suzuki, Yutaka Ohtake und Takashi Michikawa: *Boundary Smoothing for Mesh Segmentation.* International Journal of CAD/CAM, 10(1):11–19, 2010.

[ZT10] Youyi Zheng und Chiew-Lan Tai: *Mech Decomposition with Cross-Boundary Brushes.* In: *Computer Graphics Forum (In Proc. of Eurographics 2010)*, Band 29, Seiten 527–535, 2010.

[ZTS02] Emanoil Zuckerberger, Ayellet Tal und Shymon Shlafman: *Polyhedral surface decomposition with applications.* Computers & Graphics, 26(5):733 – 743, 2002.

[ZWC+10] Juyong Zhang, Chunlin Wu, Jianfei Cai, Jianmin Zheng und Xue-cheng Tai: *Mesh Snapping: Robust Interactive Mesh Cutting Using Fast Geodesic Curvature Flow.* In: *Computer Graphics Forum (In Proc. of Eurographics 2010)*, Band 29, Seiten 517–526, 2010.

Index