Studienarbeit

Plausible Reparatur von Löchern in Dreiecksnetzen

Steffen Ernst

Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik, Institut für Simulation und Graphik
Betreuer: Prof. Dr.-Ing. Bernhard Preim

angefertigt am: Konrad-Zuse-Zentrum für Informationstechnik Berlin
Betreuer: Dr. Stefan Zachow

1.10.2008 bis 31.03.2009

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben.

Magdeburg, den 31.08.2009

Steffen Ernst

Inhaltsverzeichnis

Zusammenfassung

Bei der Arbeit mit dreidimensionalen Modellen am Rechner steht man nicht selten vor der Aufgabe Löcher zu schließen. Löcher treten häufig in Oberflächen auf, die aus Aufnahmen eines Laserscanners rekonstruiert wurden. Hier ist das Schließen ein notwendiger Schritt in der Nachbearbeitung, um eine vollständige Oberfläche zu erhalten. Löcher können auch das Resultat einer manuellen Reparatur sein, etwa wenn fehlerhafte Geometrie entfernt wurde. Eine geschlossenen Oberfläche vereinfacht den Umgang mit dem Modell, sie entspricht mehr der Realität und unserer natürlichen Wahrnehmung. Löcher sind bei der Präsentation vor Publikum meist nicht hinnehmbar und die weitere Nutzung ist mit Löchern beschränkt. Viele Verfahren sind auf solche Modelle nicht anwendbar.

Uns Menschen fällt es meist leicht, uns die fehlende Geometrie vorzustellen. Auch wenn wir das Original nicht gesehen haben ist es uns möglich, mittels unserer Erfahrung eine Vorhersage über das wahrscheinliche Aussehen zu machen. Der Rechner hingegen kann im allgemeinen nicht auf so großes Wissen zurückgreifen, trotzdem ist eine plausible, unseren Vorstellungen entsprechende Reparatur wünschenswert. Ein Ziel dieser Arbeit ist es, einige ausgewählte Verfahren vorzustellen und sie hinsichtlich ihrer Plausibilität zu untersuchen. Das zweite Ziel ist die Implementierung eines dieser Verfahren und ein abschließender Vergleich der damit erzeugten Ergebnisse.

Anlass dieser Studienarbeit war mein sechsmonatiges Betriebspraktikum vom 1.10.2008 bis 31.3.2009 am Konrad-Zuse-Zentrum für Informationstechnik Berlin - kurz ZIB. Zur Visualisierung und Verarbeitung verschiedener Daten arbeitet man am ZIB zusammen mit anderen Partnern an der Software ZIBAmira®.

1 Einführung

1.1 Hintergrund

Mit Laserscannern lässt sich die Oberfläche existierender physikalischer Objekte in den Rechner übertragen. Sie tasten die Oberfläche mit einem Laserstrahl ab und erzeugen so eine Punktwolke bestehend aus dreidimensionalen Punkten. Aus diesen kann dann die Oberfläche rekonstruiert werden.

Das Problem dabei ist, dass von einem Standpunkt aus nicht die gesamte Oberfläche eingesehen werden kann. Auch durch die Kombination mehrere Ansichten aus verschiedenen Perspektiven ist häufig nicht die gesamte Oberfläche einzusehen, etwa wenn die Oberfläche konkave Stellen aufweist. Es können nur Punkte auf der Oberfläche gemessen werden, von denen ein Laserstrahl zurück zum Scanner reflektiert werden kann. Transparente Oberflächen etwa am Auge oder glänzende Flächen wie an Augen und Ohren stellen ein Problem dar, da sie den Strahl ablenken und der Scanner die Position nicht mehr exakt oder überhaupt nicht ermitteln kann. Am Rand von Objekten ist der Winkel der Oberfläche relativ zum Laserstrahl meist so schlecht, dass der Strahl nicht zum Gerät zurück reflektiert wird. Weiter Informationen können in [Bla04, BR02] gefunden werden. Löcher können auch bei der weiteren Arbeit entstehen, etwa wenn bei der Vereinfachung der Oberfläche lokal unerwünschte Topologien auftreten und diese entfernt werden.

Löcher im Sinne dieser Arbeit sind also Stellen, an denen das Innere des Modells nicht vom äußeren getrennt werden kann, das Modell ist nicht geschlossen. Sie sollten nicht mit Löchern aus dem alltäglichen Sprachgebrauch verwechselt werden: das Loch in der Mitte eines Torus ist hier also nicht gemeint, denn die Oberfläche eines Torus separiert das Innere klar vom Äußeren, sie ist geschlossen.

Der Fokus dieser Arbeit liegt auf Dreiecksnetzen und leidet sich aus der Aufgabenstellung des Praktikums ab. Da die meisten der hier vorgestellten Arbeiten direkt auf Dreiecksnetzen arbeiten können, stellt das keine wirkliche Einschränkung dar. Einige Verfahren können nur auf Punktwolken arbeiten, welche aber aus einem Dreiecksnetz erzeugt werden können. Netze aus Vierecken oder beliebigen Polygonen können in ein Dreiecksnetz überführt werden. Dreiecksnetze werden in der Geodäsie zur Landvermessung genutzt, dienen in der Finite Elemente Simulation als eine grundlegende Datenstruktur oder werden zur approximierten Darstellung realer oder komplett virtueller Oberflächen in der Computergrafik verwendet. Einen Überblick über weitere Anwendungen bietet [Wik].

1.2 Motivation

Löcher in Modellen sind aus mehreren Gründen ein Problem. Wenn man vom Modell eines realen Objektes ausgeht, dann ist der einfachste, dass ein Loch einen Fehler darstellen, welcher im Original nicht vorhanden ist. In der realen Welt sind alle Objekte geschlossen, ihre Oberfläche umschließt klar ein Volumen. Selbst ein Blatt Papier hat eine geschlossene Oberfläche und ein Volumen. Auch ein komplett am Rechner entworfenes Modell soll meist ein reales Objekt nachahmen. Modelle können für Lehrzwecke in virtuellen Museen, für die Präsentation auf einer Verkaufsplattform oder für die Unterhaltung in einem Spiel oder einem Kinofilm erstellt worden sein. In all diesen Anwendungen stellen Löcher in der Oberfläche ein ästhetisches Problem dar. Auch für Messungen wie kürzesten Abständen auf der Oberfläche oder der Bestimmung des Volumens ist eine geschlossene Oberfläche unabdingbar.

Die von einem Laserscanner stammenden Modelle besitzen meist sehr viele Löcher, so dass eine automatisierte Reparatur der Oberfläche sinnvoll ist. Für diese Aufgabe werden in den nächsten Kapiteln mehrere Verfahren vorgestellt. An die dabei eingefügte Geometrie kann man verschiedene Anforderungen stellen. Die grundlegendste ist, dass sie das Loch komplett verschließt. Die meisten im Rahmen dieser Arbeit vorgestellten Verfahren sorgen des weiteren für einen glatten Übergang zur Umgebung. Für sehr kleine Löcher sollten diese beiden Anforderungen ausreichen. Wenn die zu rekonstruierende Fläche im Verhältnis zum gesamten Modell groß ist, können aber noch weitere Kriterien wichtig werden. Wenn die Umgebung des Loches stark gekrümmt ist, wenn sie ein feines Muster aufweist oder wenn durch das Loch die Kante eines Objektes verläuft, dann ist es wahrscheinlich, dass diese Eigenschaften auch auf der fehlenden Geometrie vorhanden waren.

Wie zuvor erwähnt entstehen Löchern bei der Aufnahme der Oberfläche unter anderem an Stellen, die der Laserscanner nicht einsehen kann, etwa weil sie von Geometrie im Vordergrund verdeckt wird. Ein Mensch, der das Objekt aus der selben Position wie der Scanner betrachtet und ebenfalls das Objekt im Hintergrund nicht sehen kann, könnte zumindest eine Vermutung über die fehlende Geometrie anstellen. Für die menschliche Wahrnehmung stellt es kein Problem dar, sich die verdeckte Hausfassade hinter einer Straßenlaterne vorzustellen. Diese von einem Menschen als wahrscheinlich angenommene Geometrie wird in dieser Arbeit als die plausible Lösung für das zu füllende Loch angesehen. Eine plausible Lösung muss nicht unbedingt die real vorhandene Geometrie wiedergeben, diese ist dem Rechner auch nicht bekannt, sie ist die für Menschen

wahrscheinlichste Vervollständigung.

In der Arbeit [BF05a] von Breckon und Fisher geht es genau darum, um die amodale Vervollständigung, also die Vervollständigung von teilweise verdeckten Objekten. Aus dem Bereich der Psychologie haben sie grundlegende Prinzipien der menschlichen Wahrnehmung zusammengestellt (nachfolgend schon auf das zu Problem des Löcherfüllens umformuliert):

- Die lokale Sicht betrachtet nur die Geometrie in der Umgebung, sie vervollständigt Konturen, Kreuzungen, Kanten.
- Die globale Sicht betrachtet das gesamte Modell, zur Vervollständigung werden vorhandene Symmetrien, Muster und Regelmäßigkeiten ausgenutzt.
- Die volumenbasierte Sicht vervollständigt unter Nutzung der beiden erstgenannten Prinzipien aus den sichtbaren Konturen Oberflächen; diese wiederum schließen Volumen ein, welche sinnvoll zu einem ganzen Volumen zusammengefügt werden. Dabei wird der Tiefeneindruck mit einbezogen, fehlende Teile eines bekannten Objektes können durch Wissen vervollständigt werden.

Auf diese drei Prinzipien wirken zwei weitere Faktoren:

- die sogenannte Scene Evidence – durch Merkmale des Objektes werden Erwartungen für die Vervollständigung gebildet
- das World Knowledge – generelle Regeln für die Vervollständigung, die auf Erfahrungen basieren

Im zweiten Teil ihrer Arbeit gehen die Autoren auf die bereits vorhandene Nutzung dieser Wahrnehmungsprinzipien im Bereich der Computergrafik ein, ein Abschnitt behandelt auch Verfahren zum Schließen von Löchern.

1.3 Zielsetzung

Im Rahmen dieser Arbeit sollen zwei Ziele verfolgt werden: die Arbeit [BF05a] soll fortgesetzt und ein Algorithmus wird implementiert werden.

Die Nachahmung und Algorithmierung der menschlichen Wahrnehmung ist interessant für das Problem der Oberflächenreparatur. Damit wird es möglich, unbekannte Geometrie auf eine sinnvolle und für Menschen plausible Weise zu approximieren. Zur Vertiefung der Arbeit von

Breckon und Fisher werden in nachfolgenden Kapiteln einige der von ihnen herausgestellten Arbeiten genauer vorgestellt und untersucht. Die Auswahl wird durch weitere Verfahren aus einem Algorithmenüberblick von Ju ergänzt. Aus seiner Arbeit [Ju09] wurden weitere Vorgehensweisen ausgewählt, die eine plausible Reparatur versprechen.

Im Rahmen meines Praktikums am ZIB sollte die dort genutzte Software ZIBAmira® erweitert werden. Im zweiten Teil dieser Arbeit soll dazu einer der im ersten Teil behandelten Algorithmen implementiert werden. ZIBAmira® bietet bereits die Auswahl zwischen verschiedenen Verfahren, diese bringen aber nicht immer die gewünschten Ergebnisse. Um die Probleme der vorhandenen Lösungen besser zu verstehen, werden sie mit in den Algorithmenvergleich aufgenommen. ZIBAmira® nutzt zur Oberflächenrepräsentation Dreiecksnetze, wodurch sich der Fokus dieser Arbeit erklärt.

Der gewählte Algorithmus wird über ein schnell und einfach zu bedienendes Interface dem Nutzer zugänglich gemacht. Es soll die Möglichkeit geben, quasi per Knopfdruck alle Löcher einer Oberfläche zu füllen. Im Falle einer nicht zufriedenstellenden Lösung sollen aber auch Möglichkeiten zum manuellen Eingriff gegeben sein. Den hier erzielten Ergebnissen werden zum Abschluss der Implementierung Ergebnisse vergleichbarer Produkte gegenübergestellt. Dabei wird es nicht um die algorithmische Umsetzung, sondern rein um die Möglichkeit gehen, schnell und einfach Löcher einer gegebenen Oberfläche zu reparieren. Da nicht jede Software eine genaue Beschreibung der eingesetzten Algorithmen, noch den Quellcode selbst zur Verfügung stellt, kann ein Vergleich der verschiedenen Implementierungen hier nicht erfolgen. Neben der Gegenüberstellung der fertigen Geometrien soll auch auf die benötigte Berechnungszeit und die vorgefundene Benutzungsoberfläche eingegangen werden.

Hier nocheinmal eine Zusammenfassung der umgesetzten Ziele dieser Arbeit:

- in den zwei Nachfolgenden Kapiteln werden insgesamt 21 Algorithmen vorgestellt, drei vorhandene und vier neue werden im Detail besprochen, darauf folgt ein Überblick über die verbleibenden 14 Algorithmen
- alle vorgestellten Algorithmen werden hinsichtlich der Plausibilität ihrer erzeugten Oberflächen verglichen
- ein Algorithmus wurde implementiert und erweitert
- den vier nun vorhandenen Algorithmen in ZIBAmira® werden die Ergebnisse acht anderer Programme gegenübergestellt

2 Sieben Verfahren im Detail

In diesem Kapitel werden die ersten sieben Algorithmen detailliert besprochen. Aus ihnen wird am Ende dieses Kapitels einer für die Implementierung ausgewählt. Die ersten drei sind die bereits vorhanden Verfahren in ZIBAmira®. Nach ihrer Beschreibung wird geklärt, warum diese nicht immer optimale Ergebnisse liefern und warum eine Neuimplementierung nötig ist. Darauf folgt die Darlegung von vier neuen Algorithmen.

In dieser Arbeit werden zusammen mit dem nächsten Kapitel 18 neue Verfahren behandelt, welche hinsichtlich ihrer zu erwartenden plausiblen Ergebnisse ausgewählt wurden. Bei ihrer Sichtung wurde klar, dass eine detaillierte Besprechung aller innerhalb der zeitlichen Grenzen dieser Arbeit nicht möglich ist. Deshalb wurden für dieses Kapitel vier Algorithmen für eine genauere Untersuchung ausgewählt. Dabei wurde neben dem zu erwartenden Aufwand für die Implementierung auch die allgemeine Anwendbarkeit in Betracht gezogen. Wie man im nächsten Kapitel sehen wird, liefern einige der neuen Verfahren gute Ergebnisse, sie sind aber auf gewisse Anwendungsgebiete beschränkt.

2.1 Drei vorhandene Implementierungen

Ear Cut

Dieses Verfahren [Mei75, EET89, Ebe06] verschließt ein Loch, indem es nach und nach Randdreiecke entfernt. Ein Randdreieck besteht aus drei aufeinanderfolgenden Punkten auf der Begrenzung. In jedem Schritt wird das Dreieck entfernt, für das folgende Summe maximal ist:

$\alpha * \min(Innenwinkel) + (\alpha - 1) * \beta$, mit α als Gewichtung und β als eingeschlossenem Winkel der Begrenzung.

Die Laufzeit des Verfahrens ist $\mathcal{O}(n \log n)$ zur Anzahl der Begrenzungspunkte.

Ruled Triangulator

Dieser Algorithmus verbindet entweder alle Punkte der Begrenzung mit einem gegebenen Startpunkt oder er verbindet die Punkte zwischen zwei gegebenen Punkten zickzackartig. Wenn dabei auf dem einen Kantenzug weniger Punkte vorhanden sind, werden Punkte auf Kanten eingefügt. In beiden Fällen kann es bereits in der Ebene bei konkaven Polygonen schnell zu Überschneidungen kommen. Punkte auf Kanten einzufügen stört die Topologie des Dreiecksnetzes, wenn man das anliegende Dreieck auf der

anderen Seite nicht teilt. Eine Verbesserung wäre es, Punkte auf den nächsten Kantenpunkt zu verbinden.

Die Laufzeit beider Modi ist linear zur Anzahl der Begrenzungspunkte. Durch seine hohe Geschwindigkeit eignet sich das Verfahren, den/die Punkt(e) vom Nutzer festlegen zu lassen, um möglichst wenige Überschneidungen zu erhalten.

Min Surface

Dieser auf der Arbeit [PP93] basierende Algorithmus fügt zuerst einen Punkt ins Zentrum des Loches ein und verbindet ihn entsprechend Grafik 1 mit einigen Randpunkten. Die dabei entstehenden Dreiecke werden nun solange verfeinert, bis auf den Außenkanten genügend Punkte zur Verbindung mit der Lochbegrenzung vorhanden sind. Überschüssige Punkte werden auf Kanten oder besser auf den nächsten Randpunkt verschoben. Um Überschneidungen zu reduzieren werden die Kantenlängen global minimiert. Dazu werden alle Punkt iterativ in Richtung ihrer aufsummierten Kanten verschoben. Nach und nach bilden sich Überschneidungen am Rand zurück. Meist können so aber nicht alle Überschneidungen beseitigt werden. Die Komplexität des Verfahrens ist quadratisch. Eine Interaktionsmöglichkeit wäre es, den ersten Punkt und dessen Anbindung an den Rand vom Nutzer festlegen zu lassen. Er könnte so besser auf Gegebenheiten des Loches reagieren.

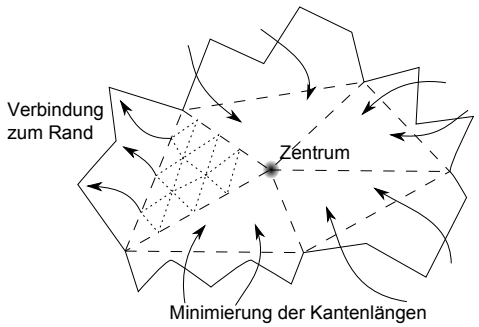

Abbildung 1: Ein Punkt im Zentrum bildet mit einigen Randpunkten Dreiecke. Diese werden wie das linke Dreieck zeigt, unterteilt, so dass am Rand neue Punkte entstehen, die mit der Originalbegrenzung verbunden werden. Durch Minimierung der Kantenlängen zieht sich die Fläche nach Innen zusammen.

Fazit

Diese drei vorhandenen Verfahren können bereits bei einfachen Löchern viele Überschneidungen erzeugen, siehe 5. Ear Cut und Ruled fügen keine

neuen Punkte ein, wodurch die eingefügten Oberflächen meist kantig und wenig plausibel wirken. Der Ruled Triangulator erzeugt auch mit Hilfe des Nutzers meist keine guten Ergebnisse. Der Min Triangulator fügt oft sehr viele Punkte ein. Durch Minimierung der Kantenlängen erreicht er eine gewissen Glättung, so dass er von den drei Algorithmen die besten Ergebnisse liefert.

Die nachfolgenden Algorithmen passen die eingefügte Oberfläche besser an die Umgebung des Loches an. Sie erzeugen dadurch weniger bis keine Überschneidungen und erzeugen eine plausiblere Geometrie als die vorhandenen Verfahren 5. Die Implementierung eines der Verfahren ist also gerechtfertigt.

2.2 Vier fortgeschrittene Verfahren

Liepa Triangulator

In [Lie03] beschreibt Liepa ein Verfahren zur Lochreparatur, was grob dem Ear Cut ähnelt, und sich entsprechend Alg. 4 aus Triangulierung, Verfeinerung und Glättung zusammensetzt.

In Alg. 5 wird die Triangulierung genauer beschrieben. Zeile 5 bestimmt die Schrittweite, welche in der ersten Runde zwei ist und Dreiecke mit aufeinanderfolgenden Randpunkten ergibt. Diese Randdreiecke bilden den Ausgangspunkt eines Pfades und werden im *W*eightset *W* verwaltet, in das sie in Zeile 12 gespeichert werden. In der zweiten Runde ist die Schrittweite drei. Für die Dreiecke gibt es nun zwei Möglichkeiten für den freien Punkt m, in Zeile 8 werden beide untersucht. An einer der beiden Kanten liegt ein Dreieck aus der ersten Runde an, gemeinsam bilden sie ein großes „Randdreieck“. Ab einer Schrittweite von vier können je nach Wahl von m an bis zu zwei Kanten bereits eingefügte Dreiecke anliegen. Wenn die Schrittweite in Zeile 5 zu groß wird hat man das Loch vollständig verschlossen. Aus *W* kann ausgehend vom zuletzt eingefügten Dreieck rekursiv bestimmt werden, welche Dreiecke kombiniert wurden. Somit erhält man alle Dreiecke eines Pfades.

In Zeile 9 Alg. 5, wird zu jedem Dreieck ein Gewicht bestehend aus der Fläche und dem Winkel zu den ein oder zwei anliegenden vorhandenen Dreiecken gespeichert. In Zeile 10 und 11 werden die Gewichte der Nachbarn noch dazu addiert. Die Flächen werden dabei addiert, bei den Winkeln wird das Maximum behalten. Der freie Punkt m wird immer so gewählt, dass der Winkel des neuen Dreiecks minimal ist. Sollte der Winkel gleich sein, wird eine kleinere Fläche bevorzugt. Damit wird eine Oberfläche mit möglichst glatten Übergängen zwischen den Dreiecken und zum

Rand erzeugt.

Entsprechend Alg. 6 werden die so gefundenen Dreiecke durch Einfügen eines neuen Punktes in der Mitte iterativ unterteilt. Nach jeder Iteration erfolgt ein Edge Swap (siehe [BB06]) entsprechend Abb. 30. Die Unterteilung wird durchgeführt, bis jedes Dreieck etwa die Größe seiner Nachbarn hat. Der Edge Swap verhindert, dass dabei lange degenerierte Dreiecke entstehen und sorgt so dafür, dass die Unterteilung beendet werden kann.

Den Abschluss bildet entsprechend Alg. 9 und Abb. 31 eine Glättung der neuen Punkte. Die Laufzeit der Triangulierung bestimmt die Geamtkomplexität und liegt entsprechend Abb. 2 bei $\mathcal{O}(n^3)$.

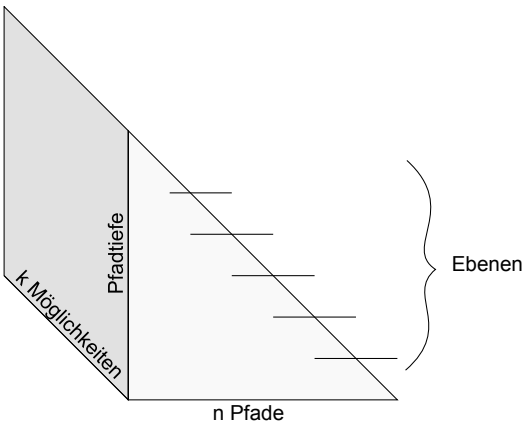

Abbildung 2: Ein Loch mit n Punkten hat nach dem ersten Schritt n offene Pfade. Nach jeder Runde gibt es auf der nächsten Ebene einen Pfad weniger. Um das Loch zu verschließen sind $n-2$ Dreiecke nötig, was der Pfadtiefe entspricht. Die Auswahl des freien Punktes m entspricht den k Möglichkeiten. k konvergiert gegen n. Gesamt: #n Pfade $*$ Pfadtiefe $*$ k Möglichkeiten $= n^3$

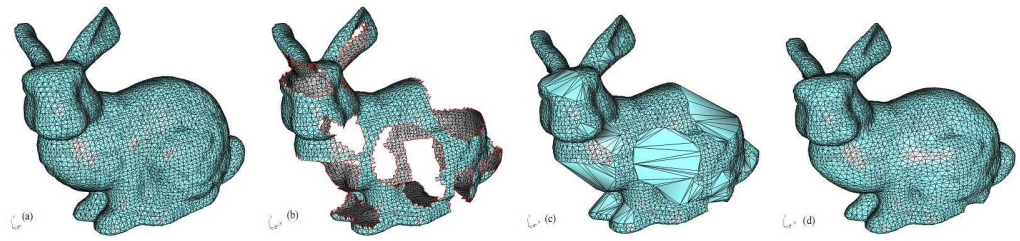

Abbildung 3: Ein Ergebnis dieses Verfahrens: das Originalmodell links wurde mit Löchern versehen, diese wurden im dritten Bild von links Trianguliert. Rechts ist das Ergebnis nach Verfeinerung und Glättung zu sehen. Quelle:[Lie03]

Volumetric Diffusion

Davis et al. beschreiben in [DMGL02] ein Verfahren, was auf einer Volumendarstellung des Modells arbeitet. Diese kann direkt aus den Ergebnissen eines Laserscanners oder indirekt aus einem Dreiecksgitter erzeugt werden.

Entsprechend Alg. 1 werden die Voxel eines Volumens mit ihrer Entfernung zur Oberfläche initialisiert; innen und außen wird anhand des Vorzeichens unterschieden. Das Zero Set besteht aus Voxeln, an denen das Vorzeichen wechselt; sie definieren die Oberfläche.

In der Nähe von Löchern wird die Distanzfunktion auf angrenzende Voxel ausgedehnt, bis das Zero Set und damit das Loch geschlossen ist. An dieser Stelle können Angaben über leeren Raum einbezogen werden, um so das Wachstum der Oberfläche zu steuern. Die Erweiterung oder Diffusion der Distanzfunktion wird durch Blurring erreicht. Dafür wird unter anderem eine $3 * 3 * 3$ Box als Faltungskern vorgeschlagen. Um die Originalgeometrie zu erhalten, wird die erweiterte Distanzfunktion mit ihrem Original kombiniert. Am Ende wird das nun geschlossene Zero Set mittels dem Marching-Cubes Verfahren extrahiert.

Eingabe : Oberfläche O

1 initialisiere Voxelgitter V mit Distanzfunktion der Oberfläche O
2 Volumengitter $Org \leftarrow V$
3 **while** *Zero set ist offen* **do**
4 erweitere die Distanzfunktion durch Blurring des Volumengitters V
5 kombiniere V mit Org
6 **end**
7 extrahiere Oberfläche mittels Marching-Cubes

Algorithmus 1 : Ablauf Volumetric Diffusion

Mit diesem Vorgehen wird eine überschneidungsfreie Oberfläche erzeugt, bei der auch Inseln automatisch mit einbezogen werden. Eine bereits als Dreiecksnetz vorliegende Oberfläche kann sich wegen der nötigen Extraktion des gesamten Volumens überall ändern. Die Komplexität für einen Iterationsschritt bestehend aus Blurring und Kombinieren mit dem Original beträgt $k * n^2 * m$, mit n als Voxelanzahl, m als halber Lochdurchmesser in Voxeln und k als prozentualer Anteil des Loches am Gesamtgitter. Die Voxelanzahl hängt von der nötigen Auflösung des Gitters ab, um alle Details einer Oberfläche wiederzugeben. Im Worst Case ist die

Komplexität eines Schrittes $\mathcal{O}(n^3)$. Die Gesamtkomplexität setzt sich aus der initialen Berechnung der Distanzfunktion, dem iterativen Verschließen des Loches und der anschließenden Extraktion der Oberfläche zusammen. Angaben dazu werden nicht explizit aufgeführt. Wegen dem hohen Speicherverbrauch wird eine Unterteilung für detaillierte, speicherintensive Modelle vorgeschlagen.

Abbildung 4: Links ein Modell mit Loch. Oben wurde es ohne, unten mit Informationen über leeren Raum verschlossen. Die Volumen mit der initialisierten Distanzfunktion werden in der Mitte gezeigt, leerer Raum ist blau markiert. Ganz rechts sind je die Volumen mit geschlossenem Zero Set (rot) zu sehen. Die schwarzen und weißen Bereiche entsprechen den Vorzeichen der Distanzfunktion. Quelle: [DMGL02]

Atomic Volumes

Podolak et al. beschreiben in [PR05] wie man Dreiecksnetze mittels Graph Cuts verschließt. Alg. 2 beschreibt den Ablauf des Verfahrens. Dabei wird über der Bounding Box ein Octree aufgebaut. Grundvoraussetzung dafür ist, dass die Oberflächennormalen global konsistent sind. Ein Gegenbeispiel ist das Möbiusband, bei dem diese Eigenschaft nicht gegeben ist und das Verfahren damit nicht funktioniert. Aus dem Octree wird ein Graph erzeugt, welcher dann mittels eines Min Cut in Innen und Außen getrennt wird. Mit den Kanten des Graphen sind Dreiecke verbunden, die Dreiecke der geschnittenen Kanten schließen so das Loch. Viele der eingefügten Flächen liegen auf den Seiten der Würfel des Octree. Um das Ergebnis ansprechender zu gestalten wird am Ende eine Glättung auf den neu eingefügten Punkten durchgeführt.

Mit diesem Vorgehen bleibt die Originalgeometrie erhalten. Für die neu eingefügten Flächen wird garantiert, dass sie die Oberfläche wasserdicht verschließen und dass es keinerlei Überschneidungen gibt. Außer durch die Wahl der Kantengewichte kann man das Ergebnis auch durch das Setzen von Punkten beeinflussen, welche innen oder außen liegen sollen, siehe Abb. 5 rechts.

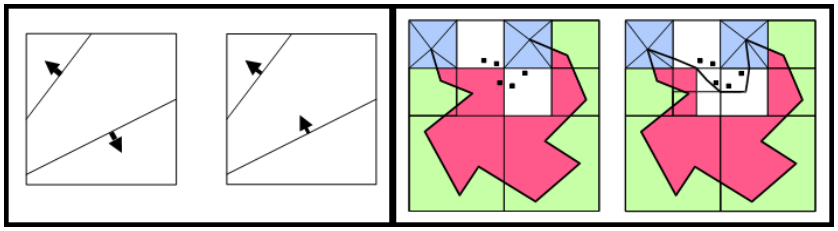

Abbildung 5: Linkes Bild: Die linke Zelle muss nicht unterteilt werden, aber die rechte, weil die mittlere Fläche anhand der Normalen auf beiden Seiten der Oberfläche liegt. Rechtes Bild: Die schwarzen Punkte wurden vom Nutzer gesetzt, sie sollen außen liegen, rechts ist die daraus resultierende Oberfläche zu sehen. Quelle:[PR05]

Die Autoren machen keine Angaben zur Komplexität ihres Verfahrens. Wenn man annimmt, dass die Glättung am Ende eine lineare Laufzeit zur Anzahl der neu eingefügten Punkte hat, dann bleibt noch der Aufbau des Octree und der Min Cut des Graphen als entscheidend für die Komplexität. Die Autoren des verwendeten Verfahrens zum Aufbau des Octree geben eine Komplexität von $\mathcal{O}(n^2)$ an, wobei n die Anzahl der Dreiecke der Ausgangsgeometrie bezeichnet. Der verwendete Algorithmus für den Min Cut hat eine Laufzeit von $\mathcal{O}(n^2 \log^3 n)$, wobei n die Anzahl der Knoten angibt. Die Knotenanzahl ist proportional zur Anzahl der Unterteilungen im Octree. Die Anzahl der Unterteilungen wiederum ist von der Lage und Größe des Loches und dem Abstand der Vertices abhängig, so dass hier keine genauere Abschätzung angegeben werden kann.

Abbildung 6: Links das Original, was in der Mitte ohne und rechts mit Nutzerangaben über leeren Raum gefüllt wurde. Quelle:[PR05]

Erweiterung der Advancing Front Method

Schilling et al. beschreiben ein Verfahren [SBSE08], was sich besonders an den Anforderungen im CAD-Bereich orientiert, indem es scharfen Kanten erhält und den Nutzer in einem interaktiven Prozess einbezieht. Die Advancing Front Method ist dafür die Grundlage, welche die Begrenzung des Loches als Front bezeichnet und diese langsam zusammenwachsen lässt.

Eingabe : Dreiecksnetz D mit konsistenten Oberflächennormalen, Funktion *getWeight* zur Erzeugung von Kantengewichten

initialisiere Octree O in der Bounding Box von D
while *Änderung von O im letzten Schritt* **do**
    foreach *Zelle Z in O* **do**
        if *Z enthält Lochbegrenzung* **then**
            if *mehr als eine Kante oder mehr als ein Vertex in Z* **then** unterteile Z
            else trianguliere Z
        end
        else
            unterteile Z, wenn Innen und Außen nicht klar getrennt (siehe Abb. 5 links)
        end
    end
end
initialisiere Graph G mit einem Source und einem Sink-Knoten
foreach *Zelle Z in O* **do**
    leere Zellen bekommen einen Knoten in G
    Zellen mit normaler Geometrie erhalten zwei Knoten in G (Innen und Außen)
    Zellen mit Lochbegrenzung erhalten für jedes bei der Triangulierung entstandene Teilvolumen einen Knoten in G
end
verbinde alle inneren Knoten in G mit Source, wichte die Kanten mit ∞
verbinde alle äußeren Knoten in G mit Sink, wichte die Kanten mit ∞
verbinde alle inneren Knoten überschneidungsfrei, ermittle die Kantengewichte mittels *getWeight*
trenne den Graph mittels eines *MinCut* in Innen und Außen, speichere die geschnittenen Kanten in P
die Füllgeometrie F für das Loch ergibt sich aus den zu P gehörenden Dreiecken
Glätte F, ohne die Struktur und damit die Topologie zu ändern

Algorithmus 2 : Ablauf Atomic Volumes

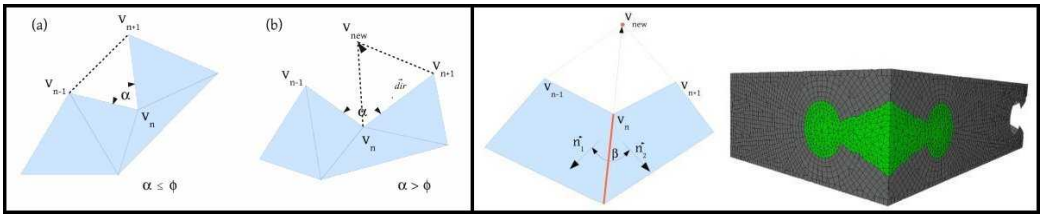

Abbildung 7: Linkes Bild: Wenn der Winkel zwischen zwei bestehenden Kanten unter einem Grenzwert liegt, werden neue Kanten eingefügt, sonst ein neuer Punkt. Rechtes Bild: Links wurde die scharfe Kante (rot) des Modells auf der rechten Seite an ihrem Winkel β erkannt. Die Kante setzt sich so auch in der neue Geometrie fort. Quelle:[SBSE08]

Am Rand der aktuellen Begrenzung werden neue Dreiecke eingefügt, wenn durch sie die Originalgeometrie nicht geschnitten wird und auch keine scharfen Kanten zerstört werden. Um Kanten zu erhalten werden neue Punkte darauf verschoben. Zur Beschleunigung der Schnitttests werden die Zäune von Schreiner et al. verwendet, Abb. 8.

Eingabe : Oberfläche O mit Lochbegrenzung L

```
1  while O bezüglich L nicht verschlossen do
2      foreach Kante K in L do
3          N ← neu einzufügende Kante oder neuer Punkt
4          if N schneidet nicht O then
5              verschiebe N auf scharfe Kanten in der Umgebung,
               falls vorhanden
6              füge N an L an, speichere sich ergebende Dreiecke D
               in O (Abb. 7 links)
7              rotiere D um K, so dass eine kurze Restbegrenzung
               entsteht
8              fasse nahe beieinanderliegende neue Dreiecke/Punkte
               zusammen
9          end
10     end
11 end
```

Algorithmus 3 : Ablauf Advancing Front

Das Verfahren sieht die Möglichkeit vor, zu Beginn eine größere Verschiebung der neuen Punkte vorzunehmen. Das hat zur Folge, dass der Übergang zur Originalgeometrie nicht mehr glatt verläuft, wie in Abb. 8 zu sehen. Auch die Anpassbarkeit der Kantenlängen ist gegeben. Sie können sich entweder an Kanten der Umgebung orientieren, oder der Nutzer

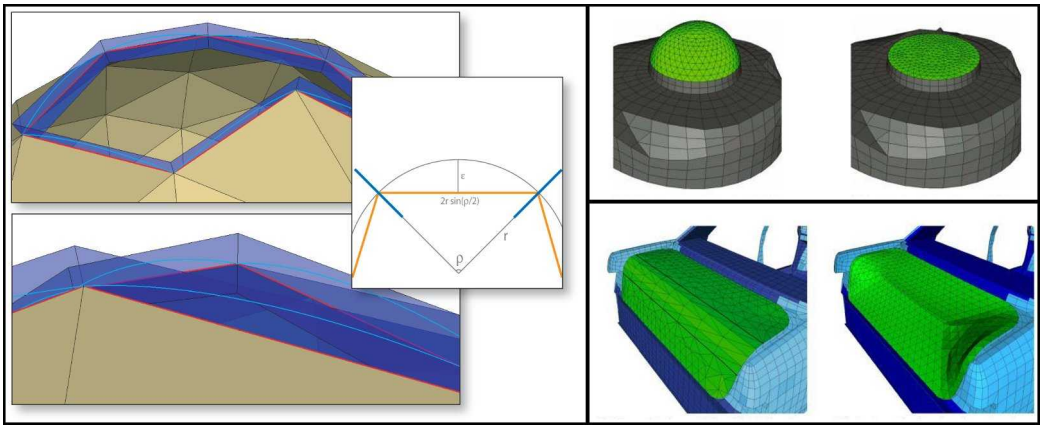

Abbildung 8: Links umgeben blaue Zäune das Loch. Sie werden anstatt der Originalgeometrie für Schnitttests verwendet. Rechts oben wurde einmal ohne und einmal mit größerer Initialverschiebung gearbeitet. Im rechten Bild wird so eine scharfe Kante als Übergang erreicht. Rechts unten: links wurde das Loch mit zwei manuell eingefügten Kanten versehen, so dass das Loch auf eine sinnvolle Weise verschlossen werden konnte. Quelle: [SBSE08]

stellt einen festen Wert ein. Letzteres verursacht aber Lücken am Rand. Das Verfahren erlaubt eine manuelle Vorunterteilung großer Löcher. Zur Vereinfachung der Interaktion werden ebene, kleine Löcher automatisch gefüllt.

Da in der Arbeit die Laufzeit nicht besprochen wird, wurde sie abgeschätzt. In jedem Schritt wird entweder eine Kante oder ein neuer Punkt eingefügt. Eine Kante verkürzt die Begrenzung um einen Punkt, entsprechend vergrößert ein eingefügter Punkt sie. Ausgehend von der Annahme, dass auf einen neuen Punkt der Fall einer neuen Kante folgt, wird die Begrenzung mit etwa jedem zweiten Schritt kleiner. Die Schrittanzahl verhält sich so linear zur Lochgröße. Nach jedem Schritt soll die eingefügte Geometrie auf Überschneidungsfreiheit mit der restlichen Begrenzung geprüft werden. Das entspricht bei n Schritten einer Komplexität von $\mathcal{O}(n^2)$. Die Anpassung der Kantenlängen wurde für diese einfache Abschätzung nicht berücksichtigt.

2.3 Vergleich

	Laufzeit	Schwierigkeit der Umsetzung[2]	erwartete Qualität*	Nuterinteraktion[3]
EarCut	++	vorhanden	-	- -
Ruled	++	vorhanden	- -	+
Min	+	vorhanden	-	- -
Liepa Triangulator	**-**	**+**	**+**	**++**
Volumetric Diffusion	- -	-	-	+
Atomic Volumes	- -	- -	+	+
Advancing Front	**+**	**+**	**+**	**++**

* siehe auch Vergleich am Ende der Arbeit
[2] basierend auf der Beschreibung im Paper (detailierte Beschreibung, Pseudocode, nötige Sekundärliteratur)
[3] wieweit kann der Nutzer das Ergebniss beeinflussen

Tabelle 1: Vergleich

In Tabelle 1 werden die Merkmale der Algorithmen auf einfache Weise gegenübergestellt. Demnach ist das Verfahren von Liepa und die Advancing Front Methode interessant für eine Implementierung. In der nachfolgenden Arbeit wird das Verfahren von Liepa umgesetzt, welches durch einen „teile und herrsche“-Ansatz beschleunigt werden soll.

3 Weitere Verfahren im Überblick

Im letzten Kapitel wurden einige Arbeiten im Detail besprochen und eine Auswahl für die Implementierung wurde getroffen. Dabei wurde ein Verfahren ausgewählt, was allgemein einsetzbar und einfach zu implementieren ist. Dafür achtet es beim Einfügen der neuen Oberfläche auf glatte Übergänge innerhalb der neuen Geometrie als auch zur Originalgeometrie. Bei größeren Löchern muss das nicht immer eine für Menschen plausible Lösung sein, etwa wenn in der Umgebung des Loches feine Muster auf der Oberfläche zu sehen sind oder eine für die Wahrnehmung wichtige Kante durch das Loch verläuft.

In diesem Kapitel werden nun 14 weiter Arbeiten behandelt, allerdings nicht so umfangreich. Die hier vorgestellten Algorithmen beziehen weitere Kriterien neben der Glattheit mit ein. Am Ende des Kapitels erfolgt eine Gegenüberstellung aller Verfahren dieser Arbeit hinsichtlich ihrer verwendeten Wahrnehmungsprinzipien, welche in der Einleitung benannt wurden und in der Arbeit von [BF05a] im Detail nachgelesen werden können.

3.1 Überblick

Robust Repair of Polygonal Models

Ju stellt in [Ju04] ein Verfahren vor, das Löcher in Polygonmodellen verschließen kann. Er wendet dabei eine Volumenmethode an, die im Unterschied zu anderen aber besonders robust zu sein scheint und scharfe Kanten erhält. Zwischen den Polygonen wird keine Konnektivität vorausgesetzt. In einem regulären dreidimensionalen Gitter werden Kanten markiert, welche von Polygonen geschnitten werden. Das Gitter wird in einem Octree gespeichert. Für die Gitterpunkte wird entsprechend der geschnittenen Kanten bestimmt, ob sie innen oder außen liegen. Dabei kann die Markierung einiger Gitterkanten getauscht werden. Gefundenen Lochbegrenzungen werden mit einer minimalen Anzahl an Gitterflächen verschlossen. Anschließend wird die nun geschlossene Oberfläche aus dem Gitter extrahiert. Wenn Normalen an den Gitterpunkten vorliegen, können auch scharfe Kanten des ursprünglichen Modells zurückgewonnen werden.

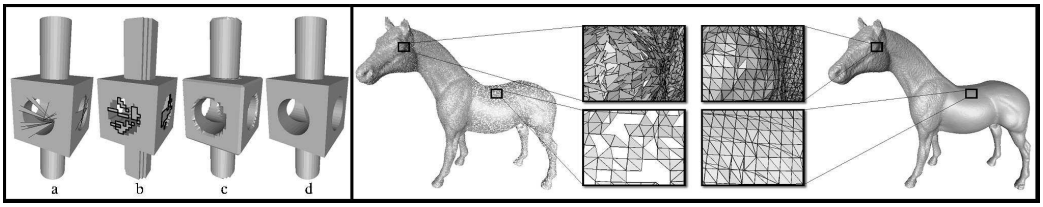

Abbildung 9: Linkes Bild: *a* zeigt ein Modell mit fehlerhaften Dreiecken, in *b* ist die Oberfläche aus dem Octree zu sehen, *c* zeigt eine Rekonstruktion des Octree mittels Marching Cube, *d* die Rekonstruktion mittels eines Verfahrens des Autors, welches die scharfen Kanten des Originals wiederherstellen kann. Rechtes Bild: Das Modell links zeigt eine künstlich gestörte Polygonoberfläche. Auf der rechten Seite die reparierte Version. Quelle: [Ju04]

Context-based Surface Completion

Das Verfahren von [SACO04] arbeitet auf einer durch Punkte repräsentierten Oberfläche, wie sie als Ergebnis eines Laserscanns entsteht. Zu jedem Punkt gehört ein Normalenvektor. Löcher sind Regionen mit einer zu geringen Punktdichte. Die Oberfläche mit seiner näheren Umgebung wird durch eine Polynomfläche niedrigen Grades approximiert. Durch rotieren, verschieben und verbiegen sollen Punktansammlungen anderer Regionen in das Loch eingepasst werden, welche eine ähnliche Charakteristik wie die Approximation aufweisen. Diese anderen Regionen stammen aus einer Trainingsmenge, in der das aktuelle Modell, sowie rotierte und gespiegelte Kopien enthalten sind. Als Erweiterung schlagen die Autoren die Integration einer Modelldatenbank vor. Die Trainingsmenge und das original Modell werden in einem Octree überführt, für die Zellen aller Ebenen wird eine Signatur berechnet. Die Signatur ergibt sich aus der Lage der in ihnen befindlichen Punkten zur Zelle, genauer gesagt aus der Lage der für diese Punkte approximierten Oberfläche. Die Signatur beschreibt die Charakteristik der in der Zelle befindlichen Punktmenge und wird für den Vergleich verwendet. Durch die Approximation der Oberfläche der Löcher kann auch für diese Zellen eine Signatur berechnet werden. In Zellen mit zu wenigen Punkten werden die Punkte aus der Zelle mit der am besten passenden Signatur eingepasst und die Zelle wird unterteilt. Das Vorgehen wird für die neuen Teilzellen wiederholt, bis sich eine geforderte Punktdichte einstellt.

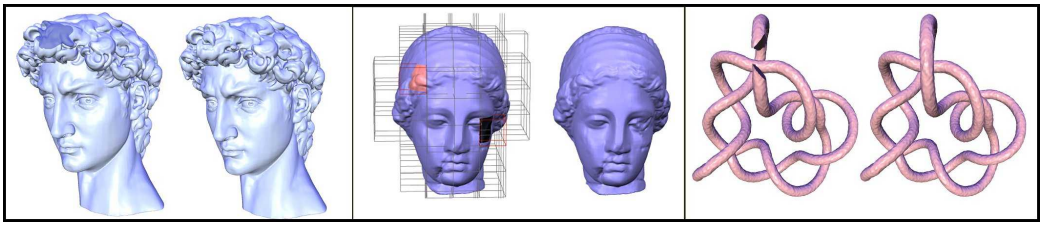

Abbildung 10: Links: Die markierte Fläche im linken Bild weist eine zu geringe Punktdichte auf, sie wird mit Geometrie aus der Umgebung gefüllt. Mitte: Geometrie mit einer passenden Signatur muss nicht die gleiche Bedeutung haben. Das Auge wurde mit Teilen vom Haar verschlossen. Solche Fälle kann der Algorithmus nicht unterscheiden. Rechtes: Auch Löcher, die nicht zusammenhängen können repariert werden. Quelle: [SACO04]

Detail-Preserving Surface Inpainting

Dieses Verfahren von Bendels [BSK05] ähnelt dem Verfahren [SACO04]; es arbeitet ebenso auf Punktdaten. Auch hier wird ein Loch iterativ mit ähnlicher Geometrie aus einer Trainingsmenge verschlossen. Wie beim Octreeansatz wird mit einer groben Approximation begonnen, die nach und nach verfeinert wird. Die gröberen Versionen der Oberfläche werden durch Smoothing erzeugt. Allen Punkten wird eine Wahrscheinlichkeit zugewiesen, ob sie korrekt positioniert sind. Originalpunkte erhalten den Maximalwert von Eins. Die Wahrscheinlichkeit wird während der Iteration ausgewertet. Dass in Schritt i mit einer gröberen Approximation verschlossene Loch wird im nächstfeineren Schritt $i + 1$ als Orientierung verwendet. In Kontrast zum Octreeansatz werden hier direkt Punkte zusammen mit ihrer jeweiligen Nachbarschaft verglichen. Der Vergleich wird nicht durch eventuell ungünstig verlaufende Zellgrenzen behindert. Während der Iteration wird für die gröber aufgelösten Modelle eine größere Nachbarschaft verwendet.

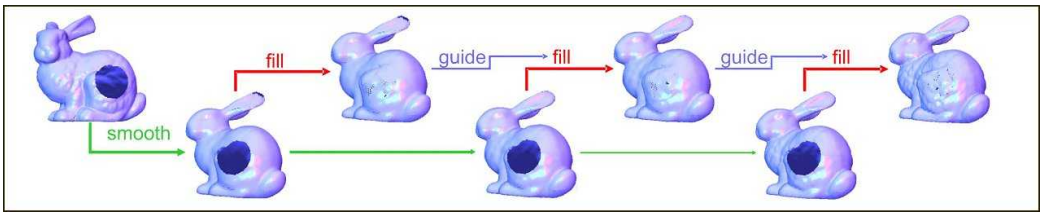

Abbildung 11: Zu Beginn werden wegen der größeren Nachbarschaft der gröberen Approximation grobe Details wie das Knie des Hasen herausgearbeitet, je feiner die Approximation der Oberfläche wird, desto feiner werden die hinzugefügten Details, die groben Details bleiben dabei erhalten. Quelle: [BSK05]

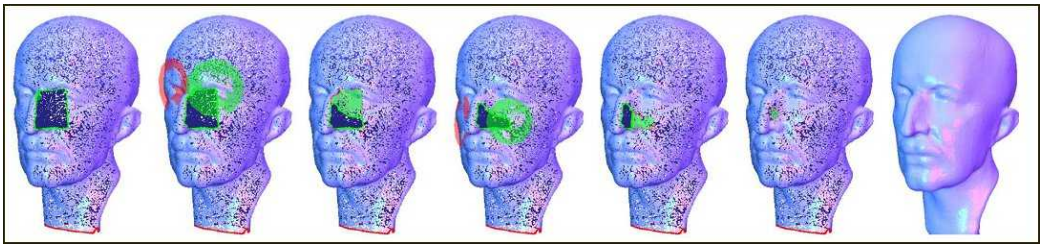

Abbildung 12: Bei sehr symmetrischen Modellen wie Gesichtern reicht teilweise ein Iterationsschritt aus. Für jeden Punkt auf der grünen Lochbegrenzung werden in der anderen Gesichtshälfte Punkte mit der bestmöglichen Übereinstimmung gesucht und so das Loch nach und nach verschlossen. Quelle: [BSK05]

Surface completion for shape and appearance

Die Arbeit von Park et al. [PGSQ06] wurde ebenfalls von [SACO04] inspiriert. Hier wird ein Octree verwendet, indem die Geometrie der Zellen anhand der Krümmung und Farbinformationen der Oberfläche verglichen werden. Löcher werden nicht durch die Punktdichte in den Zellen, sondern über die Berechnung eines Distanzfeldes gefunden. Um ein Loch zu füllen wird der Octree nach der Zelle durchsucht, die mit der Geometrie in der Umgebung des Loches am besten übereinstimmt. Wie bei [SACO04] wird die Geometrie dieser Zelle dann in das Loch eingepasst. Dabei werden auch die Farbinformationen übernommen. Falls die Automatik versagt, kann der Nutzer unter mehreren möglichen Übereinstimmungen wählen oder die zu verwendende Geometrie auf der Oberfläche markieren.

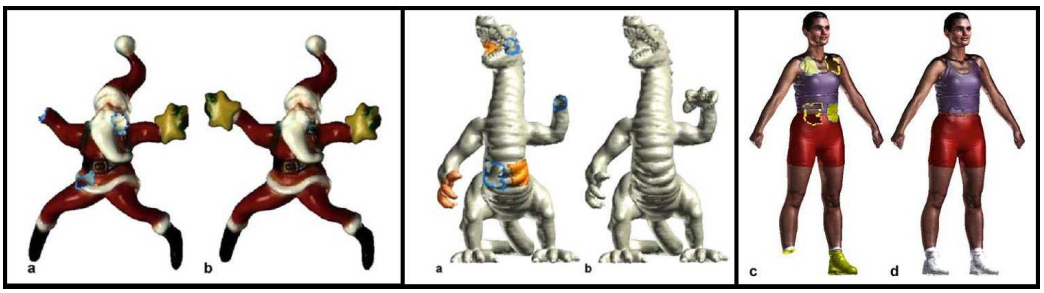

Abbildung 13: Drei Modelle mit blau umrandeten Löchern, welche automatisch gefunden wurden. Die gelben Flächen wurden vom Nutzer als Regionen zur Vervollständigung der Löcher markiert. Quelle: [PGSQ06]

Geometry Completion and Detail Generation by Texture Synthesis

Nguyen et. al. [NYC05] arbeitet auf einem Dreiecksnetz. Die dreidimensionale Umgebung eines Loches wird durch Parametrisierung der Oberfläche

in den $\mathbb{R}^2$ übertragen; Inseln können dabei integriert werden. Die Dreiecke der Oberfläche werden in ein Geometriebild gezeichnet, die Eckpunkte liegen auf den 2D-Koordinaten der Parametrisierung, die Farben ergeben sich aus den 3D-Koordinaten. Auf diesem Bild der Parametrisierung werden relativ zum Normalenvektor lokale Gradienten berechnet, welche Änderungen der Geometrie widerspiegeln. Diese Änderungen werden dann in die Bereiche des Loches mittels Textursynthese fortgesetzt. Aus den so vervollständigten Gradientenbildern wird das komplette Geometriebild erzeugt, was wiederum zur Erzeugung des endgültigen Dreiecksnetzes verwendet wird. Die neuen Punkte werden dabei zum Teil aus markanten Punkten des Geometriebildes erzeugt, weitere Punkte entstehen bei der Triangulierung dieser Punkte zusammen mit der Lochbegrenzung. In einem iterativen Prozess werden die Details des Geometriebildes auf die Triangulierung übertragen. Die neue Oberfläche enthält die Details des Geometriebildes und besteht aus Dreiecken, deren Größe sich an umgebenden Originaldreiecken orientiert.

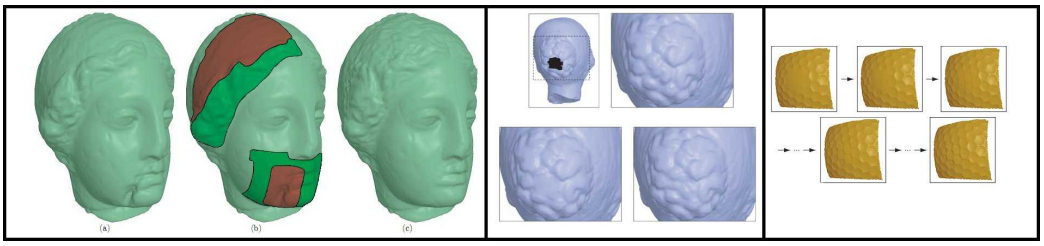

Abbildung 14: Linkes Bild: Das Verfahren eignet sich auch, um Details von Teilen der Oberfläche auf andere Gebiete zu übertragen. Hier werden Details aus den roten in die grünen Gebiete übertragen. Mittleres Bild: Ein Loch im Hinterkopf einer Statue, das Bild oben rechts zeigt die Originalgeometrie. Unten links ist die Triangulierung und unten rechts das Ergebnis zu sehen, nachdem auf diese iterativ die Details der Umgebung übertragen wurden. Rechtes Bild: Der iterative Prozess am Beispiel eines Golfballs. Quelle: [NYC05]

Non-parametric 3D Surface Completion

Breckon und Fisher gehen in [BF05b] von einer vorhandenen Triangulation eines fehlenden Bereiches aus, zum Beispiel erzeugt durch einen anderen Triangulator. Auf diese Oberfläche übertragen sie dann ähnlich wie [NYC05] Details von der Originalgeometrie. Dazu bestimmen sie zuerst eine grobe Version der Originalgeometrie plus einer Verschiebungsgeometrie, die zusammen mit der groben Version wieder das Original ergibt. In der Verschiebungsgeometrie sind die Details enthalten. Ähnlich der Textursynthese im 2D-Fall setzen sie die so extrahierten Details angefangen

vom Rand nach innen in die neue Geometrie fort, indem sie die Nachbarschaft jedes neuen Punktes mit der Nachbarschaft jedes Originalpunktes hinsichtlich seiner Verschiebungsgeometrie vergleichen. Aus den zehn Punkten mit der besten Übereinstimmung wird per Zufall einer ausgewählt dessen Verschiebung auf den neuen Punkt addiert wird.

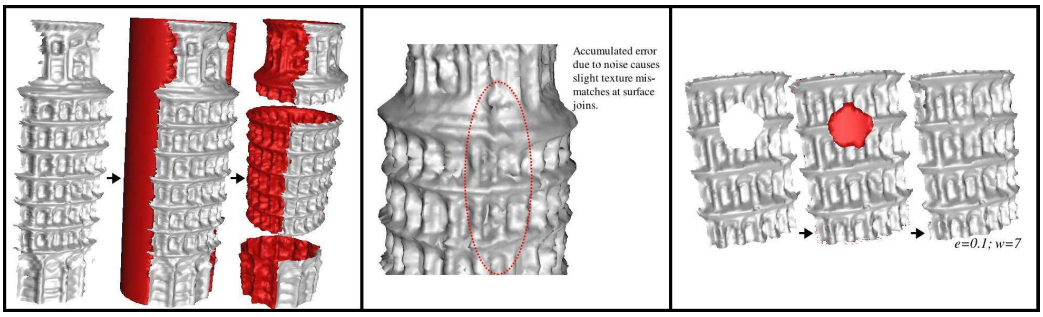

Abbildung 15: Der Turm von Pisa: linkes Bild: Nur die Vorderseite wurde aufgenommen, die restliche Geometrie wurde grob erzeugt und die Details übertragen. Mittleres Bild: Die Rückseite des Turmes weist in Folge von Rauschen Fehler an der Verbindung der beiden umlaufenden Vervollständigungen auf. Rechtes Bild: Das Verfahren wurde auf ein kleines Loch angewendet. Quelle: [BF05b]

Image Guided Geometry Inference

Xu et al. verbinden in [XGR+06] die Triangulierung eines Laserscanners mit normalen Fotos dieses Modells. Die Position der Kamera muss dabei mit dem Laserscanner abgestimmt sein; alternativ können die Fotos auch per Hand aligniert werden. Die Geometrie des Scanners wird nun in die Fotos projiziert. Anhand vollständig gescannter Oberflächenabschnitte wird eine Beziehung zwischen den Oberflächennormalen und der Beleuchtung in den Fotos hergestellt. Diese Beziehung wird genutzt, um mittels der Fotos die Geometrie in nicht eingescannten Bereichen zu rekonstruieren. Aus der Beziehung der vorhandenen Normalen am Rand und dem Winkel zur Kamera werden die Normalen im Inneren interpoliert. Bereiche, die die Kamera schlecht einsehen konnte und Bereiche mit großen Höhenunterschieden werden verworfen. Am Ende wird das Foto verwendet, was für den größten Bereich des Loches Normalen in diffus beleuchteten Bereichen enthält. In Bereichen, die im Schatten liegen oder die Highlights beinhalten ist die Normalenabschätzung zu unsicher. Aus dem gewählten Foto werden die Normalen zurück in die Scanngeometrie übertragen, wo mit ihnen eine Oberfläche aufgebaut wird.

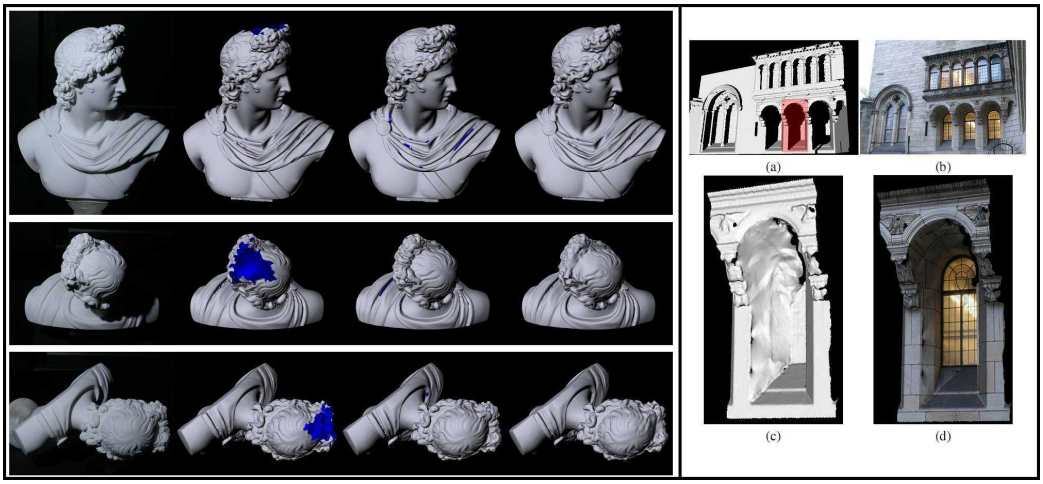

Abbildung 16: Linkes Bild: In den Zeilen sind drei Ansichten eines Modells zu sehen, was unter kontrollierten Lichtbedinungen in einem Innenraum aufgenommen wurde. Links ein Foto, Mitte links zeigt das gescannte Modell mit (blauen) Löchern, im Bild mitte rechts wurden die Löcher zum Vergleich mit neu aufgenommenen Samplepunkten verschlossen, rechts wurden die Löcher mit Hilfe der Fotos verschlossen. Rechtes Bild: Zu sehen ist in der oberen Zeile links eine gescannte Häuserfront und rechts das zugehörige Foto, in der unteren Zeile wird nur ein Häusereingang betrachtet, der linke Bereich des Eingangs konnte von der Scannerposition nicht eingesehen werden und wurde aus dem Foto rekonstruiert. Quelle: [XGR+06]

Discovering Structural Regularity in 3D Geometry

Pauly et al. nutzen in [PMW+08] die Regularität vieler Modelle aus. Aus Punkt- und Polygonoberflächen extrahieren sie ohne Vorwissen Wiederholungsregeln einzelner Elemente. Die Wiederholung kann aus Verschiebung, Rotation oder Skalierung oder einer Kombination dieser bestehen. Ziel ist das Finden möglichst kleiner Teile des Modells, die mit möglichst häufiger Wiederholung große Teile des Modells rekonstruieren können. Dazu zerlegen sie das Modell in kleine Teile, welche sie nach ihrer Ähnlichkeit gruppieren. Die Ähnlichkeit ist dabei invariant für Verschiebung, Translation und Rotation. Zur Regelfindung werden nur Modelle innerhalb einer Gruppe herangezogen. Je mehr Wiederholungen ein Teil des Modells aufweist, desto sicherer kann die dazugehörige Regel extrahiert werden. Nachdem die Regel gefunden ist kann sie zum Vervollständigen des Modells verwendet werden. Weitere Anwendungen sind die Kompression von Modellen oder die Manipulation von Modellen durch Ändern der Regel oder Austausch der zu einer Regel gehörenden Geometrie.

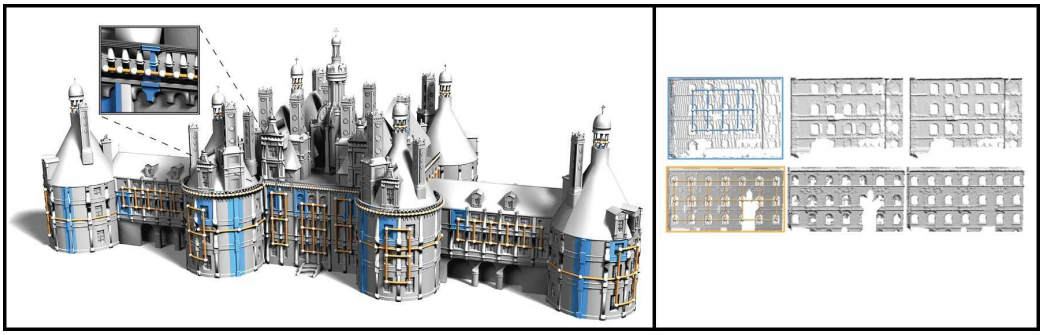

Abbildung 17: Linkes Bild: Sich wiederholende Elemente auf einem Gebäude sind in Blau gekennzeichnet. Die gelben Verbindungen beschreiben die Wiederholungsregeln, welche mit ihrem Verfahren gefunden wurden. Rechtes Bild: In der linken Spalte sind zwei per Laserscanner aufgenommene Häuserwände zu sehen. Auf ihnen hat das Verfahren die durch blaue und gelbe Verbindungen darstellten Wiederholungen gefunden. Mit dieser Regel war es möglich, die Oberflächenrekonstruktionen in der rechten Spalte zu erzeugen, in denen viele Löcher beseitigt werden konnten. Zum Vergleich sind in der mittleren Spalte Ergebnisse einer herkömmlichen Rekonstruktion zu sehen. Quelle: [PMW+08]

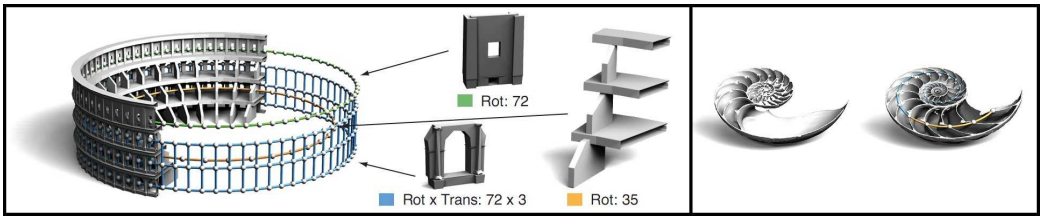

Abbildung 18: Linkes Bild: Sich wiederholende Elemente an dem Modell eines Amphitheaters. Blaue Verbindungen im Modell beschreiben die Wiederholung des Bogenelementes, das auf drei Ebenen je 72-mal rotiert wird. Der obere grüne Kreis beschreibt die Wiederholung des kleinen Fensters. Insgesamt 35-mal wird das Element im inneren bestehend aus Tragwänden und Zwischenböden auf dem gelben Kreis rotiert. Rechtes Bild: Ein Muschelmodell, was nach der Regelfindung um weitere Segmente ergänzt werden konnte. Quelle: [PMW+08]

Head shop: Generating animated head models with anatomical structure

Kähler et al. geht es in ihrer Arbeit [KHYS02] um die Erzeugung von Modellen von Köpfen, die sich für die physikalische Animation von Gesichtern eignen. In diesem Gebiet kann auf einen großen Datenbestand digitalisierter Köpfe zurückgegriffen werden, aus welchen ein Referenzmodell erzeugt wurde. Dieses generische Modell wird an die Punktwolke eines Laserscanners angeglichen, wobei automatisch fehlende Teile in der Punktwolke vom Referenzmodell übernommen werden. In der eingegebenen Punktwolke müssen manuell einige Landmarken gesetzt werden,

welche Dank des großen Vorwissens über die anatomische Beschaffenheit von Köpfen und Gesichtern automatisch weiter verfeinert werden. Dieser Ansatz ist auch auf andere Gruppen von Modellen übertragbar, solange es ein Referenzmodell gibt.

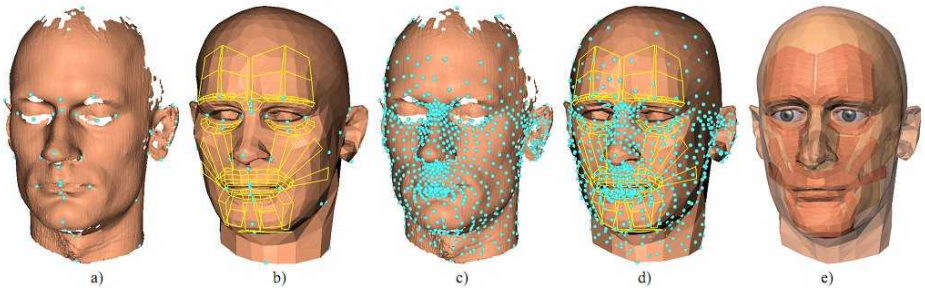

Abbildung 19: *a* zeigt Scanndaten mit platzierten Landmarken, *b* das Referenzmodell mit angedeuteten Muskeln (in gelb) nach dem ersten Anpassungsschritt an die Scanndaten, *c* die Landmarken auf dem eingegebenen Modell in *a* wurden verfeinert, *d* das Referenzmodell wurde an die neuen Landmarken angepasst, *e* das Endergebnis, vorbereitet für die Animation. Quelle: [KHYS02]

The space of human body shapes: reconstruction and parameterization from range scans

Allen et al. passen in ihrer Arbeit [ACP03] ein hoch aufgelöstes Referenzmodell eines kompletten Menschen an die Punktwolke eines per Laserscanner komplett aufgenommenen Menschen an. Beim Einpassen achten sie auf drei Kriterien: die auf den Modellen vorhandenen Marker müssen mit denen des Referenzmodells übereinstimmen. Beim Einpassen sollte sich das Referenzmodell lokal gleichmäßig ändern und die endgültige Abweichung zum Originaldatensatz muss so gering wie möglich sein. Bei Löchern wird die Abweichung ignoriert, so dass hier die Oberfläche der Referenz verwendet werden kann. Bei der Anpassung wird auch die Verlässlichkeit der Eingabedaten einbezogen. Die Autoren verwenden Modelle aus dem CESAR-Projekt, bei dem alle Messpunkte einen Konfidenzwert tragen. Der Nutzer kann Bereiche des Originalmodells auch gezielt durch das Template ersetzen lassen, wenn er beispielsweise weiß, dass das Ohr im Original schlecht repräsentiert wird.

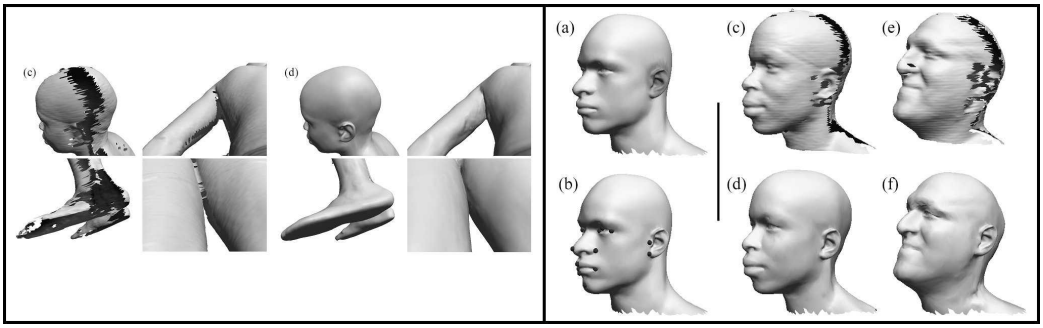

Abbildung 20: Linkes Bild: Nachdem das Referenzmodell eingepasst wurde, sind die Löcher verschlossen. Rechtes Bild: Das Ohr des Referenzmodells wurde komplett übernommen. Quelle: [ACP03]

A Statistical Method for Robust 3D Surface Reconstruction from Sparse Data

Blanz et al. vervollständigen in ihrer Arbeit [BMVS04] Modelle, die nur durch wenige Punkte gegeben sind, mit Hilfe eines Morphable Model, einer Vektorraumrepräsentation von Trainingsmodellen. In ihrer Arbeit verwenden sie die Vervollständigung von Gesichtern und von Zähnen als Beispiel. Das Verfahren ist auf andere Probleme anwendbar, wo eine Lösung linear aus gegebenen Beispielen abgeleitet werden kann. Die Eingabepunkte eines zu rekonstruierenden Modells können gewichtet werden, ihre gegebene Orientierung im Raum ist für das Verfahren unwichtig.

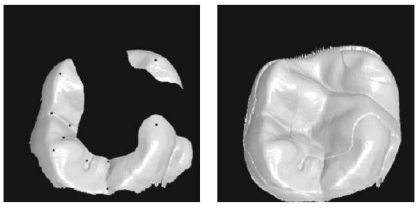

Abbildung 21: Mit einigen manuell gesetzten Merkmalspunkten können die fehlenden Teile des linken Modells mit statistischen Daten nachgebildet werden (rechts). Quelle: [BMVS04]

SCAPE: Shape Completion and Animation of People

Anguelov et al. stellen in [ASK+05] ein System namens SCAPE vor, welches anhand vieler Laserscannaufnahmen einer Person in verschiedenen Positionen auf die Änderungen der Muskulatur dieser Testposition in den einzelnen Posen trainiert werden kann. Mit diesen Trainingsdaten ist es möglich, die Veränderungen der Muskulatur dieser Person auch für noch

unbekannte Posen zu approximieren, sowie diese Muskulaturänderungen auf die Modelle anderer Personen zu übertragen. Für ein unvollständiges Modell eines Menschen können die fehlenden Teile approximiert werden. Dazu wird das Modell durch einige manuell gesetzte Korrespondenzpunkte mit dem Referenzmodell in Deckung gebracht. Aus dem Referenzmodell wird dann die Pose abgeschätzt und für diese eine entsprechende Muskulaturänderung berechnet. In einem iterativen Prozess wird die Pose und die Muskulatur weiter angepasst, bis sich aus dem Referenzmodell die fehlende Geometrie für das eingegebene Modell ergibt.

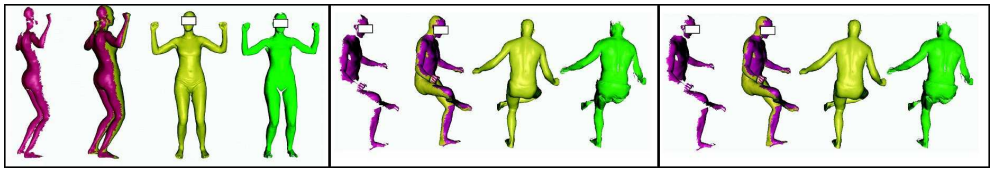

Abbildung 22: In den Kästen sind jeweils links unvollständige Aufnahmen von Menschen in verschiedenen Positionen zu sehen. Die linke Person ist in den Trainingsdaten enthalten, aber die Position ist neu, die beiden anderen Personen sowie ihre Positionen sind nicht in den Testdaten enthalten. Im zweiten Bild jedes Kastens ist die neue Geometrie in gelb gekennzeichnet. Im dritten Bild kann die Approximation mit dem Original verglichen werden, was im jeweils vierten Bild zu sehen ist. Quelle: [ASK+05]

Template-Based Mesh Completion

Kraevoy et al. stellen in [KS05] ein Verfahren vor, dass Geometrie eines Referenzmodells auf ein Modell mit Löchern übertragen kann. Die Referenz muss dabei nur die selbe Topologie wie die Eingabe aufweisen. Es ist auch möglich, vorhandene Geometrie zwischen zwei Modellen auszutauschen, die beide Löcher aufweisen, etwa wenn Objekte aus zwei verschiedenen Winkeln aufgenommen wurden und so Löcher an unterschiedlichen Positionen aufweisen. Geometrie, die in beiden Modellen fehlt, wird durch virtuelle Dreiecke erzeugt. Virtuelle Dreiecke entstehen während der Abbildung des einen Modells auf das andere und werden für die neu entwickelte Parametrisierung benötigt. Sie erlauben eine globale Parametrisierung, die alle Löcher im Modell verschließt und eine schneller ablaufende lokale Parametrisierung.

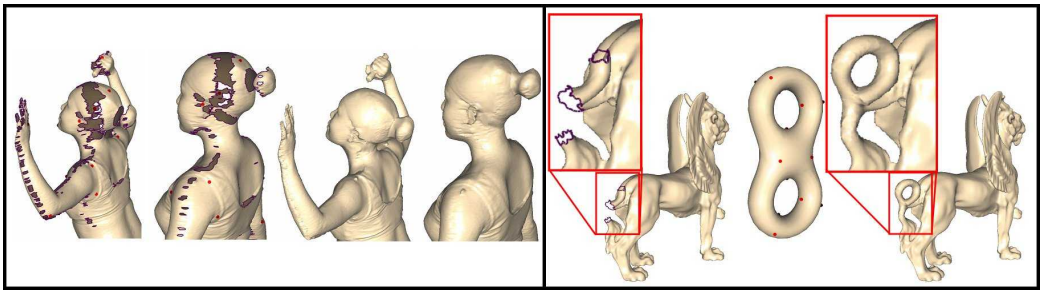

Abbildung 23: Linkes Bild: Die Geometrie zweier Modelle eines Objektes wurde genutzt, um beide zu vervollständigen, teilweise war das Einfügen virtueller Dreiecke nötig. Rechtes Bild: Der fehlende Schwanz einer Statue wurde durch das Referenzmodell (Mitte) repariert. Quelle: [KS05]

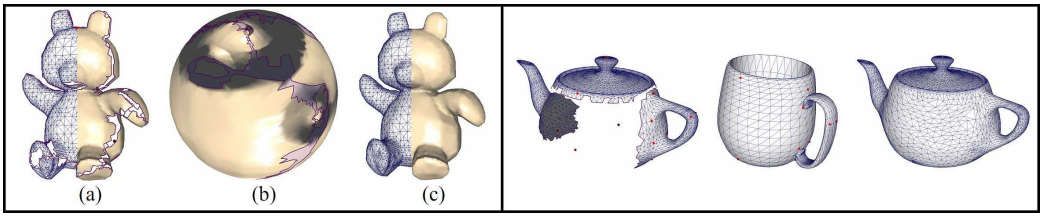

Abbildung 24: Linkes Bild: Das Modell eines Teddys wurde mit einer Kugel als Referenz vervollständigt. Rechtes Bild: Eine Teekanne wurde mit einem Becher als Referenzmodell repariert. Quelle: [KS05]

Example-Based 3D Scan Completion

Pauly et al. verwenden in [PMG+05] auch eine Datenbank mit Referenzmodellen. Allerdings wählen sie aus dieser mehrere Modelle aus, wenn diese lokale Charakteristika des eingegebenen Modells besser als andere vervollständigen können. Die Suche in der Datenbank wird durch vom Nutzer eingefügt Kontextinformationen vereinfacht. Vervollständigte Modelle werden der Datenbank hinzugefügt. Auch Zwischenergebnisse eines Modells können als Ausschnitt eingefügt werden und direkt für die Reparatur anderer Teile des selben Modells verwendet werden. Problemstellen auf dem fertigen Modell werden markiert, so dass der Nutzer gegebenenfalls eingreifen kann.

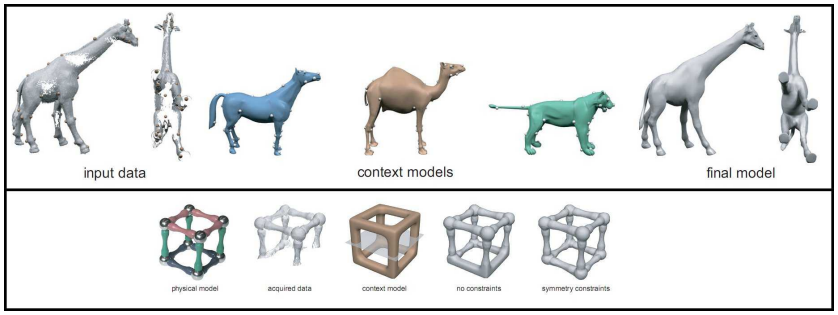

Abbildung 25: Oben: Die Modelle von drei anderen Tieren werden verwendet, um die Giraffen zu vervollständigen. Unten: Mittels Symmetrieebenen kann der Nutzer die Reparatur steuern. Quelle: [PMG+05]

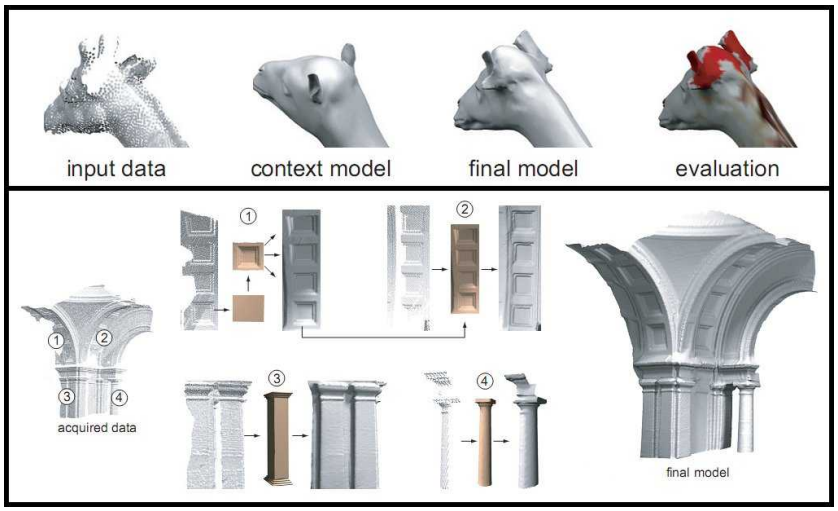

Abbildung 26: Oben: Die vom Verfahren am Ende rot markierten Bereiche zeigen im Ergebnis, wo es Probleme gab. Unten: Die aufwendige Verkleidung einer Säule wird in mehrere Teile zerlegt, Teil Eins kann nach seiner Vervollständigung für Teil Zwei als Referenz verwendet werden. Quelle: [PMG+05]

3.2 Vergleich

Hier sollen alle in dieser Arbeit besprochenen Verfahren hinsichtlich ihrer angewendeten Wahrnehmungsprinzipien (siehe 1.2) in Tabelle 2 verglichen werden.

Die Mehrzahl der gezeigten Verfahren nutzen die volumenbasierte Sicht, indem sie fehlende Geometrie mittels Wissen über ähnliche Modelle vervollständigen. Etwa die Hälfte davon kann fehlende Geometrie auch vom aktuellen Modell übernehmen, sie nutzen damit auch die globale Sicht. In vier Arbeiten wird mittels der lokalen Sicht die umgebende Geometrie in das Loch fortgesetzt, ein Verfahren achtet dabei speziell auf die Erhaltung

	lokale Sicht	*globale Sicht*	*volumenbasierte S.*	*Sonstiges*
Liepa Triangulator	Dreiecksgröße der Umgebung fortgesetzt			Ausgleich der Krümmung
Volumetric Diffusion	glatte Fortsetzung der Umgebung			
Atomic Volumes				glatte überschneidungs-freie Oberfläche
Erweiterung der Advancing Front Method	Fortsetzung der Umgebung, Kanten werden erhalten			
Robust Repair of Polygonal Models				Entfernung fehlerhafter Dreiecke
Context-based Surface Completion		Ähnlichkeit innerhalb des M.	Ähnlichkeit zu anderen M.	
Detail-Preserving Surface Inpainting		Ähnlichkeit innerhalb des M.	Ähnlichkeit zu anderen M.	
Surface completion for shape and appearance		Ähnlichkeit innerhalb des M.	Ähnlichkeit zu anderen M.	
Geometry Completion and Detail Generation ...	Fortsetzung der Umgebung			
Non-parametric 3D Surface Completion	Fortsetzung der Umgebung	Ähnlichkeit innerhalb des M.		
Image Guided Geometry Inference			Foto als "Wissen" zur Volumenvervollst.	
Discovering Structural Regularity ...		Regelmäßigkeit		
Head shop			Ähnlichkeit zu anderen M.	
The space of human body shapes			Ähnlichkeit zu anderen M.	
A Statistical Method ...			Ähnlichkeit zu anderen M.	
SCAPE			Ähnlichkeit zu anderen M.	
Template-Based Mesh Completion			Ähnlichkeit zu anderen M.	
Example-Based 3D Scan Completion		Ähnlichkeit innerhalb des M.	Ähnlichkeit zu anderen M.	

Tabelle 2: Vergleich aller neuen Algorithmen

von Kanten, zwei übertragen Details der Umgebung mit in das Loch. In [PMW+08] werden entsprechend der globalen Sicht Regelmäßigkeiten in einem Modell gesucht, um fehlende Teile zu rekonstruieren, es eignet sich besonders für Gebäude. Der in dieser Arbeit implementierte Algorithmus von Liepa gleicht die Größe der neuen Dreiecke an die der Umgebung an, was hier zur lokalen Sicht gezählt wird.

Bei einigen Verfahren fällt die starke Spezialisierung auf: [ASK+05, ACP03, KHYS02] arbeiten nur auf Ganzkörpermodellen oder Modellen von Gesichtern. Bei vielen anderen auf Ähnlichkeit basierenden Verfahren muss zumindest das richtige Modell ausgewählt werden oder die Modelldatenbank entsprechend vorbereitet sein, zumindest wenn die Löcher größer sind und innerhalb des gegebenen Modells keine Vervollständigung gefunden werden kann. Für Laserscanner ist die Arbeit von [XGR+06] interessant, bei der auf das Koordinatensystem des Scanners kalibrierte Fotos als zusätzliche Wissensquelle dienen.

Die Arbeit [PR05] nutzt selbst keines der Wahrnehmungsprinzipien. Den Bildern im Paper nach ist es aber auch zu plausiblen Ergebnissen in der Lage und erlaubt Eingriffe durch den Nutzer.

4 Umsetzung

4.1 Amira®

Amira®wurde ursprünglich in der Abteilung Visualisierung und Datenanalyse am Zuse-Institut Berlin entwickelt. Von Visage Imaging und der Visualization Sciences Group (unter dem Namen Avizo®) wird es kommerziell weiterentwickelt und vertrieben. Der Abteilung Visualisierung und Datenanalyse dient die Software als Entwicklungsplattform für die meisten ihrer Forschungsprojekte. Vom Zuse-Institut wird ZIBAmira im Rahmen von Forschungskooperationen weiterentwickelt und an Kooperationspartner gegeben.

Amira®ist durch eine Vielzahl unterstützter Datenformate, durch seine Möglichkeiten der Bearbeitung und Darstellung dieser Daten flexibel einsetzbar. In der Gruppe für Medizinische Planung am Zuse-Institut wird es zusammen mit externen Partnern für die Lösung klinisch relevanter Problemstellungen verwendet. Es wird unter andern für die Diagnose und Therapieplanung bei der Kiefer- und Gesichtschirurgie und die Segmentierung medizinischer Bilddaten verwendet.

Amira®und ZIBAmira®werden mit der Sprache C++ entwickelt und besitzt eine saubere, einfach zugängliche Codebasis. Sie sind modular Aufgebaut, die Software und der Quellcode sind umfangreich dokumentiert. Dadurch wurde eine gute Einarbeitung in die Software und die schnelle Umsetzung der Erweiterung gefördert.

Im Funktionsumfang der Software sind bereits drei Verfahren zum Verschließen von Löchern vorhanden. Diese wurden im Algorithmenvergleich dieser Arbeit behandelt. Ziel war die Integration eines weiteren Verfahrens, was besonders bei komplexen Lochgeometrien eine bessere Qualität erreicht. Die drei vorhandenen Algorithmen sind durch ein Framework in die Software integriert, welches die Suche nach Löchern in einer Oberfläche übernimmt. Es erlaubt die Auswahl eines Verfahrens, sowie die Wahl eines Loches, falls die Oberfläche mehrere Löcher besitzt. An den gewählten Lochfüller übergibt das Framework neben der Oberfläche die Begrenzung des gewählten Loches. Eine Undo-Funktion ist vorhanden, allerdings nur für das letzte verschlossene Loch. Das Framework wurde dahingehend erweitert, dass innerhalb einer Oberfläche jedes Loch wieder geöffnet werden kann. Ebenfalls wurde die Möglichkeit implementiert, alle Löcher mit einem Knopfdruck zu verschließen. In diesem Zusammenhang ist die Erweiterung der Undo-Funktion besonders hilfreich, da man so in der Nachkontrolle jedes beliebige Loch zurücksetzen kann, um es dann mit anderen Parametern oder einem anderen Verfahren zu

verschließen. Auf dieser leicht erweiterten Grundlage konnte nun mit der Implementierung des neuen Verfahrens begonnen werden.

4.2 Verschließen eines Loches

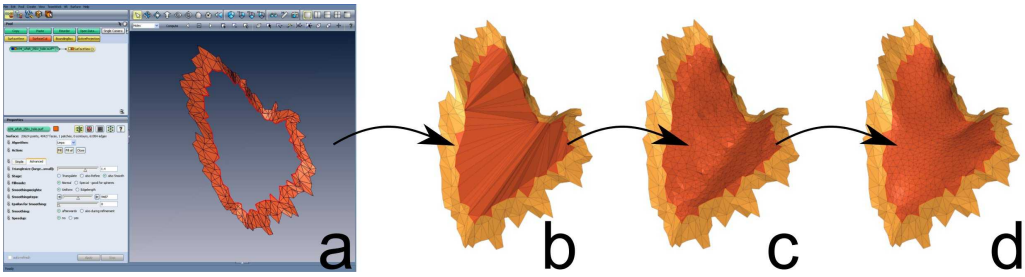

Abbildung 27: Ablauf des Algorithmus, nachdem der Nutzer in *a* ein Loch ausgewählt hat, in *b* wird die initiale Triangulierung gezeigt, in *c* sind Verfeinerung und in *d* die Glättung zu sehen

Das Vorgehen für das Verschließen eines Loches wird in der vorstehenden Grafik illustriert. Der Nutzer wählt ein Loch aus und startet das Verfahren. Das Framework liefert die Oberfläche und die Begrenzung des Loches an den Algorithmus. Dieser verschließt das Loch dann in drei Schritten. Zuerst wird eine initiale Triangulierung erzeugt, im zweiten Schritt wird diese verfeinert und anschließend werden die so neu entstandenen Punkte geglättet.

Eingabe : Eine Oberfläche S mit eine Lochbegrenzung H
Ausgabe : Eine Oberfläche P, die das Loch verschließt

1 erzeuge leere Oberfläche P
2 orientiere H passend zu den Dreiecken in S
3 $P \leftarrow$ trianguliere (S, H)
4 $P \leftarrow$ verfeinere $(P, size(H))$
5 $P \leftarrow$ glätte (P, S, H)
6 **return** P

Algorithmus 4 : Grober Ablauf des Verfahrens

Triangulation

Wie im Algorithmenvergleich bereits erwähnt verhält sich der Triangulierungsschritt ähnlich einem Ear-Cut Algorithmus. Das Loch wird auch nach und nach durch Abschneiden einzelner Dreiecke verkleinert. Bis zum

Eingabe : Eine Oberfläche S mit eine Lochbegrenzung H, Materialindex mi der Umgebung, Füllmodus f
Ausgabe : Eine Oberfläche P, die die initiale Triangulierung enthält

```
1  erzeuge passendes Weightset W für H
2  for p ← 0 to size (H)−1 do
3      W_{p,p+1} ← leeres Gewicht mit entsprechendem Dreieck aus mi
4  end
5  for stepsize ← 2 to size (H)−1 do
6      for p ← 0 to size (H)−1 − stepsize do
7          k ← p + stepsize
8          for m ← p + 1 to k − 1 do
9              w ← getWeight (p, m, k, f)
10             addiere Gewicht W_{p,m} zu w
11             addiere Gewicht W_{m,k} zu w
12             if w < W_{p,k} then W_{p,k} ← (w, m)
13         end
14     end
15 end
16 erzeuge leere Oberfläche P
17 P ← Dreiecke aus W beginnend mit W_{0,size(H)−1}
```

Algorithmus 5 : Ablauf des ersten Schrittes, der Triangulation

Einfügen des letzten Dreiecks werden allerdings mehrere Wege verfolgt. Für die Verwaltung dieser Wege wird ein *Weightset* verwendet. Es ordnet jedem offenen Weg ein Gewicht zu. Das *Weightset* beginnt wie in Zeile 3 Alg. 5 zu sehen mit einem leeren Gewicht für jede Kante. Diese Gewichte bilden die n Startpfade in Abb. 2, wenn die Lochbegrenzung aus n Kanten besteht. Sie werden in der untersten Ebene gespeichert.

In einer Schleife werden jetzt Dreiecke für alle möglichen Schrittweiten eingefügt, beginnend mit einer Schrittweite von zwei, das Maximum liegt bei $n - 1$. Für jede Schrittweite existiert eine Ebene im *Weightset* und die Kanten der untersten Ebene entsprechen damit einer Schrittweite von eins. In Zeile 6 und 7 aus 5 wird mit p der erste und mit k der dritte Punkt des aktuellen Dreiecks festgelegt. Die Schleife mit Beginn in Zeile 8 legt den zweiten Punkt m fest, welcher zwischen p und k liegt. Ab einer Schrittweite von drei gibt es für den mittleren Punkt m mehrere Möglichkeiten, jede Möglichkeit entspricht einem Dreieck. Für jedes dieser Dreiecke wird ein Gewicht bestimmt. Das kleinste Gewicht wird zusammen mit Punkt m in das *Weightset* eingefügt.

Das letzte Dreieck beginnt mit Punkt $Null$ und endet mit Punkt $n - 1$, der mittlere Punkt m ist mit im Gewicht gespeichert und so kann dieses Dreieck einfach aus dem *Weightset* in die neue Oberfläche kopiert werden. Nun werden rekursiv die Dreiecke zur linken und rechten bestimmt und ebenfalls in die Oberfläche eingefügt. Das sich diese so einfach finden lassen liegt daran, dass sich die Gewichte über den ersten und dritten Punkt ihres Dreiecks ansprechen lassen. Im ersten Rekursionsschritt muss man also für das linke Dreieck zum Gewicht mit Start bei $Null$ gehen, das Ende liegt bei m.

Offen ist noch, wie das Gewicht für ein Dreieck aufgebaut ist und wie es bestimmt wird. Der Aufbau besteht aus Winkel und Fläche. Bei einem Ear Cut hat das neue Dreieck zwei Nachbarn, weil die Kante vom ersten zum dritten Dreieckspunkt durch die Mitte des Loches verläuft. Nur bei maximaler Schrittweite liegt auch an der dritten Kante ein Dreieck an. Diese Dreiecke können Originaldreiecke sein, wenn das aktuelle Dreieck an der Begrenzung des Loches anliegt. Wenn es nicht an der Begrenzung anliegt, dann wurden die Nachbardreiecke bei einem Ear Cut mit einer kleineren Schrittweite erzeugt. In der Funktion in Zeile 9 wird zu jedem vorhandenen Nachbardreieck der Winkel bestimmt und das Maximum für das Gewicht gespeichert. Hier wird auch die Fläche des Dreiecks in das Gewicht übertragen. In den Zeilen 10 und 11 werden die Gewichte der beiden Nachbarn hinzu addiert. Der eventuelle dritte Nachbar im letzten Schritt ist eine Kante. Beim Addieren von Gewichten werden die Flächen ebenfalls addiert, bei den Winkeln wird wie zuvor nur der maximale Win-

kel übernommen. Um dann zu entscheiden, ob ein Gewicht kleiner ist, wird zuerst der Winkel betrachtet. Ein Gewicht ist kleiner, wenn es einen kleineren Winkel enthält. Nur wenn die Winkel gleich sind, wird die Fläche betrachtet und eine kleinere Fläche wird bevorzugt. Wenn zwei Pfade im *Weightset* durch Einfügen eines neuen Dreiecks zusammengefasst werden, bekommen die beiden Pfade mit dem kleinsten maximalen Winkel den Vorzug. Wenn man nur auf die Fläche achten würde könnten Zacken am Rand des Loches durch invertierte Dreiecke überlagert werden, wodurch unendlich dünne Stellen entstehen, siehe auch 28.

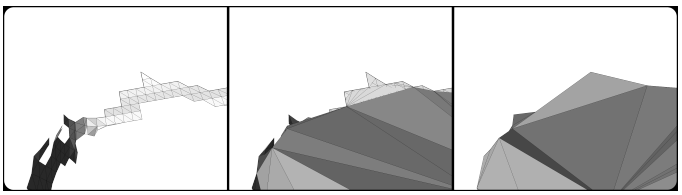

Abbildung 28: Links: Beispiel für Zacken am Rand des Loches, Mitte: ohne Berücksichtigung des Winkels, Rechts: mit dessen Berücksichtigung gefüllt.

Verfeinerung

Eingabe : Die zu verfeinernde Oberfläche S, in der die ersten b Punkte zur Begrenzung gehören, die Größe m des kleinsten angrenzenden Dreiecks im Original und ein Faktor s, der die Größe der neuen Dreiecke im Verhältnis zur Umgebung bestimmt
Ausgabe : Das Ergebnis wird direkt in S gespeichert

```
1 ordne jedem Dreieck den Zustand unverändert zu
2 ordne jedem Punkt in S den Mittelwert der Länge der
  anliegenden Begrenzungskanten als Faktor scale zu
3 while refineTriangles () do
4     swapEdges ()
5 end
```

Algorithmus 6 : Vorbereitung der Verfeinerung

Im Schritt der initiale Triangulierung wurde das Loch mit neue Dreiecken bereits vollständig verschlossen. Vor allem, wenn größere Zwischenräume zu überbrücken waren und die Begrenzung weit außerhalb einer Ebene liegt, kann die so entstandene Oberfläche sehr kantig sein. Im dritten Schritt findet deshalb eine Glättung statt. Damit diese funktioniert,

werden in diesem Schritt neue Punkte eingefügt. Die Position eines neuen Punktes ergibt sich aus dem Mittelwert der drei Punkte eines Dreiecks. Wie in Zeile 3 Alg. 6 zu sehen werden in einer Schleife abwechselnd neue Punkte in die Dreiecke eingefügt und Edge-Swaps durchgeführt. Ob ein neuer Punkt eingefügt wird ist abhängig von der Länge der umgebenden Kanten. In Zeile 2 wird für den späteren Vergleich jedem Begrenzungspunkt ein Faktor zugeordnet, der sich aus der Länge der beiden anliegenden Begrenzungskanten ergibt. Die neu eingefügten Kanten werden hier nicht berücksichtigt. Dadurch entsteht in mehreren Iterationen ähnlich große Dreiecke wie in der Umgebung des Loches.

Eingabe : Dies Funktion kann auf alle Variablen aus Alg. 6 zugreifen
Ausgabe : *true* wenn neue Punkte eingefügt wurden, sonst $false$

1 $numTriangles \leftarrow size(P)$
2 markiere jedes unbeendete Dreieck als unverändert
3 **for** $t \leftarrow 0$ **to** $numTriangles$ **do**
4 überspringe Dreiecke, die verändert oder beendet sind
5 $p_{1,2,3} \leftarrow$ *scale*-Faktor der Eckpunkte
6 $cScale \leftarrow$ Mittelwert aus p_i
7 $center \leftarrow$ Mittelwert der drei Eckpunkte
8 $dist_{1,2,3} \leftarrow$ Abstand von *center* zu den Eckpunkten
9 multipliziere jedes $dist_i$ mit s
10 **if** *wenn für mindestens ein i gilt* $dist_i > max(p_i, cScale)$ *und das Dreieck nicht sehr viel kleiner als alle anderen Dreiecke ist* **then**
11 markiere Dreiecke als beendet, deren Fläche kleiner als alle anderen inneren und angrenzenden Dreiecke ist und deren größter Innenwinkel nahe 180° ist
12 überspringe beendete Dreiecke
13 Unterteilung durch Verbinden aller Eckpunkte mit *center*
14 versuche einen Edge-Swap mit unveränderten Nachbarn
15 markiere alle geänderten und neuen Dreiecke als verändert
16 **end**
17 **end**
18 **return** $numTriangles < size(P)$

Algorithmus 7 : Ablauf von refineTriangle

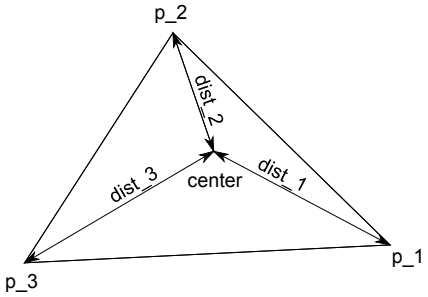

Abbildung 29: Unterteilung eines Dreiecks. Der neue Punkt *center* wird mit den drei Eckpunkten durch Kanten verbunden, wenn mind. eine Entfernung $dist_i$ größer als $max(p_i, (p_1 + p_2 + p_3)/3)$ ist.

In Funktion 7 wird für jedes Dreieck der Mittelpunkt als Mittelwert der Eckpunkte berechnet. Analog werden die zuvor ermittelten Faktoren der Randpunkte zu einem Mittelwert m zusammen gefasst. Dieser Mittelpunkt wird nur als neuer Punkt eingefügt, wenn die Entfernung zu einem Eckpunkt größer ist, als der an diesem Eckpunkt gespeicherte Faktor oder m. Wenn der Punkt eingefügt wird, dann erhält er als Faktor m. Damit werden die Kantenlängen am Rand an die neuen Punkte propagiert. Wenn auf einer Seite des Loches nur kleine Dreiecke angrenzen und auf der gegenüberliegenden Seite sehr große, dann entstehen dazwischen auf diese Weise Dreiecke, die von der einen zur anderen Seite größer werden. Wie in Zeile 9 in Alg. 7 zu sehen, kann durch einen Skalierungsfaktor Einfluss genommen werden. Wenn s größer als eins ist, dann werden die Dreiecke kleiner, umgekehrt mit einem Wert kleiner eins bleiben sie größer. Mit s kann also die Anzahl der neu eingefügten Punkte und somit auch die der Dreiecke und deren Größe angepasst werden. Der in Zeile 10 und 11 durchgeführte Vergleich der Flächen dient der Sicherstellung, dass der Algorithmus endet. In Zeile 11 werden sehr kleine spitzwinklige Dreiecke als beendet markiert, damit werden sie nicht weiter unterteilt. Ohne diese Erweiterung kommt es am Rand teilweise zu Problemen mit vielen degenerierten Dreiecken, die immer wieder unterteilt werden. Auch der nachfolgende Edge-Swap konnte das Problem allein nicht beseitigen.

In Algorithmus 8 ist das Vorgehen beim Edge-Swap zu sehen. Hier wird mehrmals hintereinander über alle Kanten iteriert, bis sich keine Veränderung mehr einstellt. Für jede Kante wird geprüft, ob einer der freien Punkte des einen Dreiecks im Umkreis des anderen Dreiecks liegt. Der Punkt, der jeweils nicht zur Kante gehört, ist der freie Punkt.

Der Edge-Swap hat zwei Auswirkungen: er verkleinert die Anzahl spitzwinkliger Dreiecke in der eingefügten Oberfläche. Dadurch werden zum zweiten die neuen Punkte gleichmäßiger über die Oberfläche verteilt. Ohne das Tauschen der Kanten würden im inneren spitzwinkliger Dreiecke

Eingabe : Dies Funktion kann auf alle Variablen aus Alg. 6 zugreifen
Ausgabe : *true* wenn Kanten geändert wurden, sonst $false$

```
1  changedSomeEdges ← true
2  markiere jedes unbeendete Dreieck als unverändert
3  while changedSomeEdges do
4      foreach Kante in S do
5          überspringe Kanten mit geänderten oder beendeten
           Dreiecken
6          if Edge-Swap ist sinnvoll und neue Kante existiert noch nicht
           in S then
7              tausche die Kante
8              markiere beide Dreiecke als geändert
9          end
10     end
11 end
12 return true wenn Kanten geändert wurden, sonst false
```

Algorithmus 8 : Ablauf von swapEdges

immer wieder spitze Dreiecke entstehen, wodurch die Punkte hier dichter liegen als in Dreiecken, die von Anfang an gleichwinklig waren. So können Regionen mit weniger dichten Punkten nach der Glättung kantig wirken. Dass spitzwinklige Dreiecke in Regionen mit starker Krümmung entstehen und so vielleicht scharfe Kanten der Umgebung fortsetzen, konnte nicht festgestellt werden. Es wurde aber auch nicht genauer untersucht. Hier liegt eine Erweiterung für zukünftige Arbeiten.

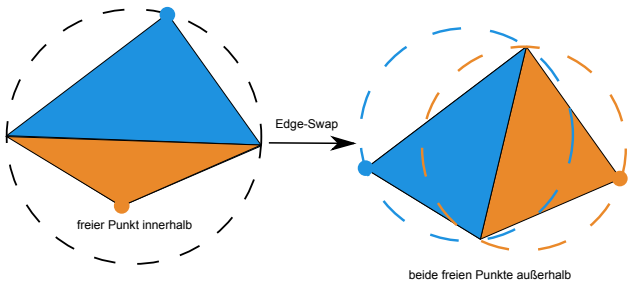

Abbildung 30: Beispiel eines Edge-Swap. Links ist der untere Punkt im Umkreis des blauen Dreiecks. Nach dem Tauschen der Kanten auf der rechten Seite liegen beide Punkte außerhalb des jeweils anderen Umkreises.

Glättung

Eingabe : Oberfläche S mit Lochbegrenzung H, Schrittanzahl n, Grenzwert für minimale Änderung e, Art m der Gewichte
Ausgabe : Oberfläche S wird direkt geändert

1 **if** m = *scaled* **then**
2 ordne jeder Kante als Gewicht das Inverse ihre Länge zu
3 **end**
4 **else**
5 ordne jeder Kante als Gewicht 1 zu
6 **end**
7 speichere für jeden Punkt die Summe der anliegenden Kantengewichte
8 **for** *step* $\leftarrow$ 0 **to** *n und Änderung im letzten Schritt größer e* **do**
9 **foreach** *Punkt p der an ein neues Dreieck grenzt* **do**
10 $nSum \leftarrow$ summiere *Nachbarpositionen/Kantengewichte*
11 $nSum \leftarrow nSum/$ Summe der anliegenden Kantengewichte
12 $pos \leftarrow$ Position des aktuellen Punktes
13 speichere Vektor $v \leftarrow (nSum - pos)$ für diesen Punkt
14 **end**
15 **foreach** *neu eingefügten Punkt in S* **do**
16 $nSum2 \leftarrow$ summiere $v/$*Kantengewicht* der Nachbarn
17 $nSum2 \leftarrow nSum2/$ Summe der anliegenden Kantengewichte
18 $nSum2 \leftarrow nSum2 - v$ des aktuellen Punktes
19 Dämpfung von $nSum2$
20 $Punktposition \leftarrow -nSum2$
21 **end**
22 **end**

Algorithmus 9 : Ablauf der Glättung

Die neu eingefügten Punkte sollen nun durch eine Laplace-Glättung so verschoben werden, dass die Winkel an den Übergängen der Dreiecke möglichst klein ausfallen. Dazu wird die nötige Krümmung zur Überbrückung eines Loches gleichmäßig an alle Kanten verteilt, indem eine Glättung zweiter Ordnung angewendet wird. In der Schleife beginnend in Zeile 9 Alg. 9 wird für jeden Punkt ein Verschiebungsvektor v berechnet, der in das Zentrum seiner Nachbarn zeigt. Durch Einbeziehen der

Kantenlänge kann der Einfluss weit entfernter Nachbarn gesteuert werden. In der Schleife ab Zeile 15 wird die Positionsänderung eines Punktes nun aus dem Mittelwert über den Vektoren v der Nachbarn minus dem eigenen Vektor v gebildet. Die Position eines Punktes bleibt unverändert, wenn er genau soweit vom Zentrum seiner Nachbarn entfernt ist, wie diese durchschnittlich von ihrem jeweiligen Zentrum entfernt sind. Durch iterative Ausführung der Glättung soll erreicht werden, dass alle Punkte gleich weit vom Zentrum ihrer Nachbarn entfernt sind, dann ist die Krümmung auf der gesamten Oberfläche optimal an alle Kanten verteilt. Zu beachten ist, dass nur innere Punkte verschoben werden, die Punkte auf der Begrenzung bleiben an ihrem Ort. Die Dämpfung in Zeile 19 bewirkt, dass der Punkt nicht um die gesamte Strecke verschoben wird. Für Kantengewichte von Eins hat sich die Multiplikation mit 0.8 als günstig erwiesen, für Kantengewichte die abhängig von der Kantenlänge sind, wurde empirisch ein Faktor von 0.25 ermittelt. Damit wird erreicht, dass die Änderung der Oberfläche langsamer stattfindet. Ohne die Dämpfung besteht die Gefahr, dass die Punkte an ihrer theoretisch optimalen Position vorbeiwandern.

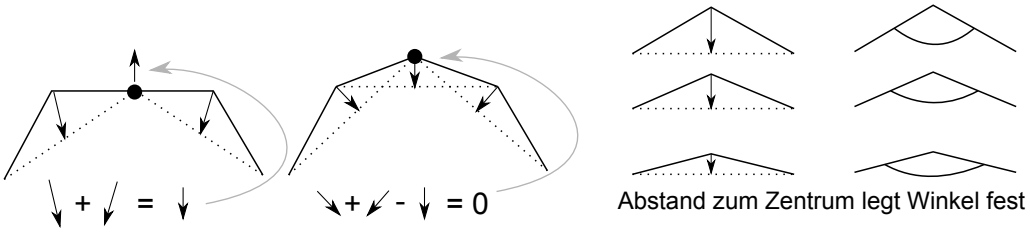

Abbildung 31: Die Glättung schematisch am 2D-Fall dargestellt. Links wurde ein neuer Punkt eingefügt, er liegt im Zentrum seiner beiden Nachbarn. Die Pfeile stellen den Vektor zum Zentrum der Nachbarn dar. Der neue Punkt liegt zu Beginn im Zentrum und hat somit keinen darstellbaren Vektor. Unter dem linken Bild wird der Mittelwert der Verschiebevektoren für den neuen Punkt gebildet. Negiert ergibt das die Verschiebung für den neuen Punkt. Das zweite Bild von links zeigt das Ergebnis dieser Verschiebung. Der neue Punkt hat nun auch einen Vektor der zum Zentrum zeigt. Die Pfeile der Nachbarn haben nun ihre Richtung geändert. Unter dem zweiten Bild muss nun die Richtung des neuen Punktes vom Mittelwert der Pfeile der Nachbarn abgezogen werden. Da der neue Punkt genau soweit vom Zentrum entfernt ist wie seine Nachbarn kann er nicht weiter verschoben werden. Die Krümmung ist nun ausgeglichen. Die beiden Bilder ganz rechts verdeutlichen den Zusammenhang zwischen dem Abstand zum Zentrum und dem Winkel für den 2D-Fall. Ein größerer Abstand ist gleichbedeutend mit einem größeren Winkel. Wenn alle Punkte den gleichen Abstand zu ihrem Zentrum haben, dann schließen die beiden anliegenden Kanten den selben Winkel ein.

4.3 Erweiterungen des Algorithmus

An dieser Stelle der Arbeit ist der Algorithmus von Liepa vollständig umgesetzt. Bei Versuchen mit dem Algorithmus sind unschöne Eigenschaften aufgefallen beziehungsweise haben sich bestätigt. Eine davon ist die kubische Abhängigkeit der Laufzeit von der Lochgröße. Wegen dieser Abhängigkeit wird sich die Laufzeit bei einer Verdoppelung der Lochgröße nicht einfach nur verdoppeln, sondern verachtfachen. Zur Verdeutlichung ein kleines Beispiel. Wenn man für das Füllen eines Loches mit 100 Punkten auf seinem Rechner eine Minute benötigt, dann wird der selbe Rechner für ein Loch mit 200 Randpunkten etwa acht Minuten brauchen.

Eine weitere unschöne Eigenschaft des Algorithmus ist, dass die erzeugte Oberfläche Überschneidungen enthalten kann. Diese können durch weit vom Loch entfernte Geometrien erzeugt werden und sind nur schwer zu vermeiden. In den meisten Fällen werden Probleme aber durch eine komplizierte Randgeometrie erzeugt und sind wesentlich leichter zu beheben.

Verbesserungspotential wurde auch bei der Glättung gesehen. Als Beispiel der Veranschaulichung soll eine Kugel mit abgeschnittenem oberen Drittel dienen. Dabei ist zu beachten, dass die Begrenzung in einer Ebene liegt. Nach Triangulierung (Schritt 1) und Verfeinerung (Schritt 2) liegen alle neuen Punkte auch in dieser Ebene. Die Glättung müsste nun alle Punkte sehr weit bewegen, um wieder eine echte Kugel zu erzeugen. Der Gedanke ist nun, dass eine einmalige Glättung nach dem ersten Verfeinerungsschritt die hier noch wenigen neuen Punkte schnell aus der Ebene des Loches heben kann. Somit lägen alle weiteren neuen Punkte ein ganzes Stück näher an ihrer Endposition. Erste Versuche dahingehend haben aber gezeigt, dass bei der Glättung während des zweiten Schritts Fehler entstehen. Die Punkte werden zu weit verschoben. Der Grund hierfür konnte bis zum Ende der Arbeit nicht mehr gefunden werden und kann in späteren Arbeiten weiter untersucht werden.

Laufzeitverbesserung

Das der Algorithmus eine kubische Laufzeit hat bedeutet nicht, dass er generell langsam ist. Gerade bei kleinen Löchern fällt dies nicht ins Gewicht und der Algorithmus ist hier relativ schnell. Wie schon mehrfach in dieser Arbeit erwähnt ist die Größe abhängig vom Rechner und auch die Geduld des Anwenders spielt eine Rolle.

Die Grundlage der Laufzeitverbesserung bildet die Beobachtung, dass die asymptotische Laufzeit bei kleinen Problemgrößen nicht zum tragen

kommt. Demnach kann es sinnvoll sein, ein großes Loch in mehrere kleinere Löcher zu zerlegen. Um bei dem anfänglichen Beispiel dieses Kapitels zu bleiben: wenn man es schafft, das doppelt so große Loch in zwei gleich große Teile zu zerlegen und beide dann separat verschließt, dann würde sich die reine Laufzeit für den Algorithmus von Liepa nicht mehr verdoppeln, sondern sie würde linear ansteigen. Ein solches Vorgehen, also das Lösen eines Problems durch Zerlegung in einfacher zu lösende Teilprobleme ist in der Informatik als teile und herrsche bekannt.

Damit diese Idee funktioniert, müssen mehrere Probleme gelöst werden. Es muss ein Verfahren zum Unterteilen eines Loches gefunden werden, was folgende Eigenschaften besitzt. Es muss schnell genug ablaufen, damit ein Laufzeitgewinn bleibt. Und die für eine Teilung eingefügte Geometrie muss innerhalb des Loches verlaufen, so dass die Originalgeometrie nicht geschnitten wird. Um das Verfahren transparent zu halten sollte der Nutzer auch die Möglichkeit haben, die Unterteilung zu kontrollieren und gegebenenfalls anzupassen. Somit hätte man quasi einen automatischen und einen halbautomatischen Modus. Zur Komplettierung kann noch ein komplett manueller Modus vorgesehen werden, für besonders schwierige Lochgeometrien. Wenn die Interaktion hierfür einfach genug ist, kann der Nutzer so eventuell noch schneller eine Unterteilung vornehmen, als es der automatische Algorithmus könnte.

Die so gefundenen Unterteilungen müssen nun noch an den eigentlichen Algorithmus von Liepa übergeben werden. Dafür kann man das Loch erst unterteilen und dann die Teillöcher unabhängig voneinander verschließen, oder man übergibt die gefundenen Trennlinien als zusätzliche Eingabe mit an den Algorithmus von Liepa.

Allgemein ist eine gültige Unterteilung eines Loches ein Kantenzug, der auf einem Punkt der Begrenzung des Loches beginnt und auf einem anderen Punkt der Begrenzung endet. Um diese Unterteilung direkt in das Dreiecksnetz zu speichern ist es nötig, die Unterteilung als Dreiecksstreifen (oder Triangle Strip) auszuführen. Auch als Eingabe für den Liepaalgorithmus ist ein solcher Streifen sinnvoll, da für die initiale Triangulierung und für den Glättungsschritt Nachbarschaftsinformationen benötigt werden. Im Sinne dieser Arbeit kann ein solcher Streifen auch nur aus einem einzigen Dreieck bestehen.

Wenn der Kantenzug aus mehr als einer Kante besteht, dann können die inneren Punkte des Kantenzuges jede beliebige Position annehmen. Eine einfache Möglichkeit ist, sie linear zwischen Anfangs- und Endpunkt zu interpoliert. Interessant ist auch die Anzahl und der Abstand dieser Punkte. Es ist beispielsweise möglich, den Abstand an dem der Begrenzungspunkte des Loches zu orientieren. Um einen Triangle Strip zu bil-

den, kann parallel zum ersten Kantenzug ein zweiter angelegt werden, bei dem mindestens ein Endpunkte um einen Punkt auf der Begrenzung versetzt wurde.

Die Frage ist nun: Wie findet man den Start- und Endpunkt einer Unterteilung? Hierfür sind viele Möglichkeiten denkbar: man kann nach kürzesten Abständen zwischen den Begrenzungspunkten suchen, oder man extrahiert scharfe Kanten auf gegenüberliegenden Seiten des Loches und versucht, diese zu verbinden. Dabei ist aber auch zu bedenken, dass Löcher nur selten in einer Ebene liegen, sie können sich im Raum winden. Daher müsste man jede gefundenen Kante gegen die Originalgeometrie auf Schnittpunkte testen. Bei größeren Löchern kann es sinnvoll sein, gleich mehrere Unterteilungen zu finden. Bei mehreren Unterteilungen müsste man wieder darauf achten, dass diese untereinander nicht zu Überschneidungen führen. Der rein automatische Modus sollte Verbindungen einfügen, die nahe der Oberfläche liegen, die ohne die Unterteilung entstanden wäre. Also nahe der Dreiecke liegen, die der Algorithmus von Liepa eingefügt hätte, wenn er auf das großen Loch angewendet worden wäre. Bei durch den Nutzer eingefügten Verbindungen kann eine anderes Aussehen der Oberfläche gewollt sein. Der automatische Modus sollte aber eine Oberfläche erzeugen, deren Charakteristik nicht vom Original abweicht. Wenn die durch die Unterteilung entstandene Oberfläche stark von der Oberfläche abweicht, die ohne sie erzeugt worden wäre, dann müsste man sich Gedanken darum machen, ob die neue Oberfläche wirklich besser ist. Die hierfür nötige Programmlogik soll hier nicht behandelt werden. Ein solcher Effekt kann im Rahmen dieser Arbeit immer noch durch manuelle Nutzereingaben herbeigeführt werden und ist eine mögliche Erweiterung zukünftiger Arbeiten.

Die Charakteristik der eingefügten Oberfläche wird im Algorithmus von Liepa maßgeblich durch die initiale Triangulation festgelegt. Es liegt nahe die Triangulierung aus dem ersten Schritt als Grundlage der Unterteilung zu verwenden. Wenn man nur die Triangulierung auf ein Loch anwendet, hat man als Ergebnis viele neue Kanten im Inneren des Loches. Jede Kante ist eine mögliche Unterteilung. Damit man auch einen Laufzeitgewinn erzielen kann, soll die Triangulierung auf einer vereinfachten Begrenzung durchgeführt werden.

Für die Vereinfachung sind wiederum mehrere Verfahren möglich: eine der einfachsten ist, dass man nur jeden n-ten Punkt der Begrenzung verwendet. Eine bessere Methode ist, dass man in Abschnitten Punkte entfernt, in denen sich die Richtung der Begrenzung nur wenig ändert. Man kann die Oberfläche auch insgesamt vereinfachen und so automatisch zu einem Loch mit weniger Randpunkten kommen. Die Triangulie-

rung angewendet auf die vereinfachte Begrenzung ergibt Kanten, die je nach Qualität der Vereinfachung mehr oder weniger nahe der gesuchten Oberfläche verlaufen. Für die Teilung werden solche Kanten heraus gesucht, die die anfängliche Begrenzung in mehrere Teillöcher zerlegt, von denen jedes unterhalb einer gewissen Punktanzahl bleibt.

Je kleiner die jeweiligen Löcher am Ende sind, desto schneller können sie befüllt werden. Allerdings besteht so die Gefahr, dass die Oberfläche stärker von ihrem Original abweicht, nämlich dann, wenn der Fehler durch die Vereinfachung der Begrenzung zu groß war und die für die Teilung verwendeten Kanten doch weiter von der Triangulierung des großen Loches abweichen. Denkbar ist auch, dass zu kleine Löcher eine sehr komplizierte Randgeometrie bilden.

In dieser Arbeit soll aus der Vielzahl der Möglichkeiten folgendes umgesetzt werden: Um den generellen Nutzen der Beschleunigung zu untersuchen wird die einfachste Möglichkeit der Vereinfachung implementiert. Die neue Begrenzung ergibt sich aus jedem n-ten Punkt. Durch Versuche soll ein geeignetes n und eine günstige Lochgröße ermittelt werden. Die Unterteilung wird als einfaches Dreieck ausgeführt, indem zu den beiden Punkten der Verbindung ein direkter Nachbar auf der Begrenzung verwendet wird. Die Unterteilung wird in den Algorithmus eingebunden, indem die neuen Teillöcher mit dem ersten Schritt trianguliert werden. Die so entstandenen Dreiecke bilden zusammen mit den Verbindungsdreiecken das Ergebnis, welches dann normal an Schritt zwei und drei übergeben wird. So werden auch die Verbindungen weiter unterteilt und zum Schluss alle Punkte gemeinsam geglättet. Die Annahme ist, dass die vorab eingefügten Verbindungen am Ende nicht mehr sichtbar sind und eine einheitliche Oberfläche als Füllung für das Loch entsteht.

In Alg. 10 ist der Ablauf der Beschleunigung zu sehen. Zu Beginn wird das Loch mittels der Funktion in 11 vereinfacht. Bei der Vereinfachung wird eine neue Oberfläche erzeugt, die nur aus Dreiecken besteht, welche an der neuen Begrenzung anliegen. Diese Dreiecke haben zwei festgelegte Punkte, die aus der Originalbegrenzung stammen. Der freie Punkt wird wie in Zeile 5 und 6 in Funktion 11 angedeutet, aus den anliegenden Dreiecken der Originalbegrenzung erzeugt. Das einfache Verfahren wird in Abb. 32 genauer erklärt.

Eingabe : Oberfläche S mit Lochbegrenzung H, gewünschte maximale Lochgröße s
Ausgabe : Dreiecke D, die das Loch vollständig verschließen

```
1  (Oberfläche P mit Lochbegrenzung L) ← simplifyHole (S, H)
2  trianguliere P
3  markiere innere Dreiecke in P als unbearbeitet
4  boundaryOfHoles ← füge ein Loch an
5  holeSize ← 0
6  queue ← init. Breitensuche mit letztem inneren Dreieck in P
7  while nicht jedes innere Dreieck bearbeitet do
8      t ← hole erstes Dreieck aus der queue
9      markiere t als bearbeitet
10     füge Punkte von t an letztes Loch in boundaryOfHoles an
11     foreach Kante e von t do
12         ++holeSize
13         if e=innere Kante then füge angrenzende, unbearbeitete
           Dreiecke an queue an
14         else holeSize ← holeSize+ Anzahl Kanten die e in H
           entspricht
15     end
16     if holeSize > s or queue ist leer then
17         boundaryOfHoles ← füge ein Loch an
18         holeSize ← 0
19         if queue ist leer then queue ← unbesuchtes innerens
           Dreieck aus P, falls vorhanden
20         else entferne außer dem Ersten alle Dreiecke aus queue
21     end
22 end
23 T ← leere Liste von Dreiecken
24 foreach Loch L in boundaryOfHoles do
25     sortiere Punkte in L, entferne Duplikate, bilde L auf H ab
26     füge weggelassene Punkte bei vereinfachten Kanten ein
27     füge Verbindungsdreieck für jede innereKante aus L in S ein
28     trianguliere L, füge entstandene Dreiecke zu T hinzu
29 end
30 return T
```

Algorithmus 10 : Ablauf des Einfügens von Verbindungen

Eingabe : Oberfläche S mit Lochbegrenzung H, Schrittweite $stepSize$
Ausgabe : Oberfläche P mit Lochbegrenzung L

1 erzeuge leere Oberfläche P
2 $p \leftarrow 0$
3 **while** $p < size(H)$ **do**
4 $\quad end \leftarrow p + stepSize$
5 $\quad T \leftarrow$ anliegende Dreiecken zwischen p und end
6 $\quad freePoint \leftarrow$ kombiniere die freien Punkte in T
7 $\quad$ bilde Dreieck t aus p, end, $freePoint$
8 $\quad$ füge t in P ein
9 $\quad p \leftarrow end$
10 **end**
11 ermittle Lochbegrenzung L aus P
12 **return** P und L

Algorithmus 11 : Ablauf von simplifyHole

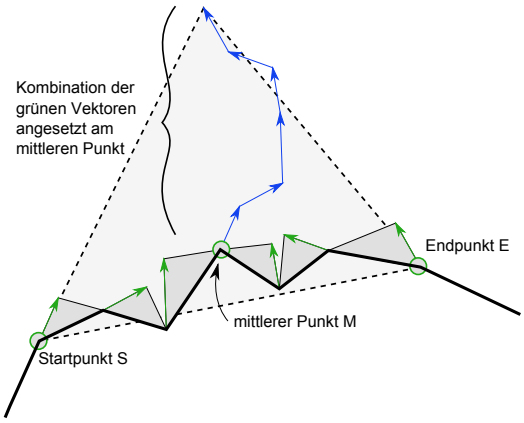

Abbildung 32: Zu sehen ist ein Abschnitt der Originalbegrenzung. Die Punkte zwischen Start- und Endpunkt werden weggelassen, um zu einer einfacheren Darstellung mit weniger Randpunkten zu kommen. Für jedes zwischen Start- und Endpunkt angrenzende Dreieck wird ein Vektor bestimmt, veranschaulicht durch die grünen Pfeile. Den Endpunkt bildet immer der freie Punkt des Dreiecks. Für den Startpunkt wird für die Dreiecke vor Punkt M der erste, und nach Punkt M der zweite Punkt der Begrenzung verwendet. Der durch Addition dieser Vektoren entstandene neue Vektor wird an Punkt M angelegt und bildet so zusammen mit Punkt S und E das neue Dreieck für die vereinfachte Lochgeometrie.

Nach der Vereinfachung wird das neue, kleinere Loch mit Alg. 5 trianguliert. In der Schleife beginnend in Zeile 7 Alg. 10 werden mittels Breitensuche zusammenhängende Dreiecksflächen gesucht. In jedem Schritt der

Schleife wird die Fläche um ein angrenzendes inneres Dreieck erweitert. Die Begrenzung dieser Flächen besteht aus Randkanten und inneren Kanten des vereinfachten Loches. Wenn man die inneren, vereinfachten Kanten auf die Originalbegrenzung abbildet, kann man die Größe einer Fläche ermitteln. Das geschieht in Zeile 14. Eine Fläche wird beendet und eine neue begonnen, wenn ihre Begrenzung größer als die gewünschte Größe geworden ist oder die Breitensuche keine Dreiecke mehr findet, weil sie am Ende eines Seitenarmes angelangt ist.

Wenn alle inneren Dreiecke von der Schleife aus Zeile 7 besucht wurden, werden die gespeicherten Ränder der Flächen in Zeile 24 auf die Originalbegrenzung abgebildet und durch die Vereinfachung ausgesparte Punkte eingefügt. Für alle inneren Kanten werden in Zeile 27 die angrenzenden Dreiecke aus der vereinfachten Triangulation eingefügt. Der Winkel zu diesem Dreieck kommt dem Winkel in der letztendlichen Oberfläche sehr nahe, wenn der Fehler der Vereinfachung gering ist. Die dadurch entstandenen Teillöcher werden normal mit Schritt eins des Algorithmus von Liepa trianguliert.

Die Schrittweite bestimmt über den Grad der Vereinfachung eines Loches und somit direkt über die Dauer der Triangulation des neuen Loches. Sie muss mindestens zwei betragen, um eine Vereinfachung und so eine Beschleunigung zu erreichen. Bei zu großer Schrittweite steigt die Wahrscheinlichkeit, dass konkave Teile der Begrenzung ignoriert werden und so Kanten zur Wahl stehen, die „außerhalb“ des Loches verlaufen. Kanten, die außerhalb liegen, erhöhen die Chance von Überschneidungen mit der Originalgeometrie. Sehr gerade verlaufende Löcher mit wenigen Zacken können mit größerer Schrittweite vereinfacht werden.

Die Triangulation des vereinfachten Loches ist nur die Vorarbeit, mit der mögliche Unterteilungen gefunden werden. Die eigentliche Beschleunigung ergibt sich aus der Zerlegung in möglichst viele kleine Löcher. Wenn die zur Unterteilung verwendeten Dreiecke aus einer nicht vereinfachten Oberfläche stammen, dann kann eine Zerlegung in beliebig kleine Dreiecke verwendet werden. Eine Vereinfachung kann Kanten enthalten, die konkave Teile oder Zacken Abb. 28 abschneiden. Sie liegen meist am Rand. Je kleiner die Lochgröße, desto höher die Wahrscheinlichkeit, dass solche Kanten als Unterteilung verwendet werden. Damit können Überlagerungen (Abb. 28 mitte) und Überschneidungen entstehen. Eine größere Lochgöße resultiert in weniger Teillöchern, deren Unterteilungen mit hoher Wahrscheinlichkeit im inneren des Loches und somit weit weg vom Rand verlaufen.

Je größer die Beschleunigung sein soll, desto mehr Teillöcher müssen entstehen und desto kleiner muss also die Größe der Teillöcher sein. In

Versuchen war es sinnvoll, ebenfalls die Schrittweite mit steigender Lochanzahl zu erhöhen, sonst nahm die Triangulation der Vereinfachung im Verhältnis zur Restlaufzeit einen zu großen Anteil ein. Für die Oberfläche der Software wurde ein Slider implementiert, mittels dem man gleichzeitig die Anzahl der Teillöcher wie auch die Schrittweite beeinflusst. Ein Schritt auf dem Slider entspricht einem Teilloch mehr, bei jedem zweiten Schritt wird ebenfalls die Schrittweite erhöht. Der Slider hat sich in Versuchen bewährt und wird für die hier gewählte einfachen Umsetzung als ausreichend angesehen.

Umgang mit Überschneidungen

Das Verfahren von Liepa reagiert in seiner Originalausführung nicht auf Überschneidungen, die die neu erzeugt Oberfläche mit sich selbst oder der Originalgeometrie erzeugt. Überschneidungen können zum einen entstehen, wenn die Originalgeometrie in das Loch hinein ragt. An so einen Fall ist der Algorithmus nur schwer anzupassen. Falls das Loch nicht durchstoßen wird sondern nur eine Spitze hinein ragt, dann könnte durch eine Anpassung der Glättung gegengesteuert werden. Die Spitze könnte als zusätzliches Gewicht dienen, um die Glättung in die entgegengesetzte Richtung zu bewegen.

Wie sich bei Versuchen mit dem Algorithmus gezeigt hat, kann es aber noch zu anderen Überschneidungen kommen. Bei sehr komplizierten Randgeometrien kann es in der Randzone zu Problemen kommen. Diese sind häufig lokal begrenzt und es sind nur wenige Dreiecke beteiligt. In Folge der Glättung kann es teilweise auch zu Falten in der Oberfläche kommen, die sich teilweise durchdringen. Diese Art von Überschneidungen soll nachfolgend durch das Entfernen von Kanten gelöst werden. Dreiecke, die Überschneidungen hervorrufen, sollen dabei durch das Entfernen von Kanten zusammenfallen. Der Punkt eines Problemdreiecks, mit den meisten anliegenden Problemdreiecken wird auf den nächstgelegenen Eckpunkt des aktuellen Dreiecks verschoben. Durch die Wahl des nächstgelegenen Punktes soll die Änderung der Oberfläche gering gehalten werden, um so neuen Überschneidungen vorzubeugen. Diese sehr einfache Methode soll als Nachbearbeitung in den Algorithmus integriert werden, um die Qualität der resultierenden Oberfläche zu verbessern.

Gewichte der Triangulierung

Bei der initialen Triangulierung werden Dreiecke auf Grundlage ihrer Winkel zu den Nachbardreiecken eingefügt. Die Fläche der Dreiecke wird nur

Eingabe : Oberfläche S mit inneren Punkten I_p und innernen Dreiecken I_t
Ausgabe : Dreiecke werden aus S entfernt, um Überschneidungen zu beseitigen

1 $P \leftarrow$ Dreiecke I_t, die andere Dreiecke aus S schneiden
2 **while** *P nicht leer* **and** *Änderung im letzten Schritt* **and** *Schrittanzahl* < 30 **do**
3 zähle für jeden Punkt in S anliegende Dreiecke aus P
4 **foreach** *t in P* **do**
5 $p \leftarrow$ Eckpunkt von t in I_p mit meisten anliegenden Dreiecken aus P
6 verschiebe p auf den nächstgelegenen Eckpunkt von t, welcher ungleich p ist
7 **end**
8 aktualisiere P
9 **end**

Algorithmus 12 : Entfernung von Überschneidungen durch Entfernung von Kanten

betrachtet, wenn die Winkel zweier Gewichte gleich sind. Im Versuch hat es sich bei manchen Lochgeometrien als nützlich erwiesen, nur die Fläche zu betrachten, wodurch eine minimale Oberfläche als Füllung für das Loch entsteht. Besonders bei sehr langgezogenen und dünnen Löchern kann damit ein sehr gutes Ergebnis erzielt werden. Auch beim Einsatz der Beschleunigung durch Unterteilung kann das außer Acht lassen des Winkels Vorteile haben. Damit muss nicht auf die virtuellen Dreiecke der vereinfachten Randgeometrie zurückgegriffen werden, womit sie als Fehlerquelle ausgeschlossen werden. Bei der Beschleunigung kann damit bei dünnen Löchern eine meist fehlerfreie Oberfläche erzeugt werden. Die Idee, die Gewichte linear zu kombinieren war nicht erfolgreich

Die Implementierung dieser Arbeit kann nicht entscheiden, wann welche Gewichte gewählt werden sollen. Deshalb wurde in der Benutzungsschnittstelle eine manuelle Wahlmöglichkeit vorgesehen. Ein Verfahren zur Automatisierung kann in späteren Arbeiten implementiert werden.

Wahl der letzten Kante

Eine letzte kleine Erweiterung betrifft die initiale Triangulierung. Wenn man sich Zeile 6 in Alg. 5 ansieht, fällt auf, dass die durch das Loch ge-

legte Diagonale nie über den Startpunkt hinausgehen kann. Es gibt also keine Dreiecke mit den Eckpunkten i, j, k für die gilt $i > k$. Theoretisch werden damit nicht alle möglichen Dreieckskombinationen getestet, die Teilmenge der Kombinationen ist vom Startpunkt abhängig. Die Implementierung wurde dahingehend erweitert und in einigen Modellen konnten damit Verbesserungen der Oberfläche beobachtet werden, siehe Abb. 34. Der Nachteil dabei ist, dass sich für die Triangulation die Laufzeit verdoppelt, aber zu Gunsten der Qualität wurde diese Erweiterung beibehalten.

5 Experimentelle Validierung

5.1 Erwartungen und Ziele

Ein Ziel dieser Arbeit war die Implementierung eines Algorithmus zum Verschließen von Löchern in aus Dreiecken bestehenden Oberflächen. Dieser Algorithmus sollte mit komplizierteren Lochgeometrien umgehen können, als die bereits vorhandenen Verfahren in ZIBAmira® und eine Oberfläche erzeugen, die sich sinnvoll in die Oberfläche einpasst. Im Idealfall soll der Nutzer nicht mehr erkennen, wo sich das Loch befunden hat.

Um die Tauglichkeit der Implementierung zu zeigen, soll sie bezüglich Bedienerfreundlichkeit, Geschwindigkeit und Qualität mit anderen Softwarelösungen verglichen werden.

5.2 Durchführung

Nutzen der Erweiterungen und Schwächen

		Beschleunigung		*Zeit*			Über-	subjektive	*Reparatur*	
	Gewichte	Schrittw.	Teile	Gesamt	Triangulation	**Faktor**	schn.	Beurteilung	**Restübers.**	Zeit
Filigree	**Winkel**	0	0	8,9	8,6	**1**	2	+	**0**	0,1
271 Punkte	**Fläche**	0	0	8,9	8,6	**1**	0	+	**0**	0,1
	Winkel	3	10	1,4	1,0	**9**	52	- -	**3**	0,5
	Fläche	3	5	1,4	1,0	**9**	0	+	**0**	0,1
	Winkel	5	10	0,7	0,3	**29**	41	-	**6**	0,7
	Fläche	5	8	0,7	0,3	**29**	0	+	**0**	0,1
Large	**Winkel**	0	0	554,2	549,1	**1**	125	+	**115**	10,4
988 Punkte	**Fläche**	0	0	512,6	501,4	**1**	1127	-	**587**	18,4
	Winkel	3	16	57,1	51,2	**11**	270	-	**184**	11,6
	Fläche	3	21	48,7	38,6	**14**	1145	- -	**596**	17,8
	Winkel	5	18	14,6	9,5	**58**	302	-	**172**	11,2
	Fläche	5	23	19,2	8,1	**68**	1114	- -	**645**	19,1
	Winkel	10	20	12,9	2,5	**220**	953	- -	**406**	19,2
	Fläche	10	26	7,8	1,8	**305**	1012	- -	**606**	10,4

Tabelle 3: Alle Verbesserungen zusammengefasst. In den Zeilen wurde immer abwechselnd Winkel- und Flächengewichte verwendet. Der Faktor beschreibt die Beschleunigung bei der Triangulierung. Die verbleibenden Restüberschneidungen sind in der vorletzten Spalte zu sehen. Zeiten sind in Sekunden angegeben. Die Spalte „subjektive Beurteilung“ gibt den visuellen Unterschied wieder, der sich durch die Beschleunigung ergibt.

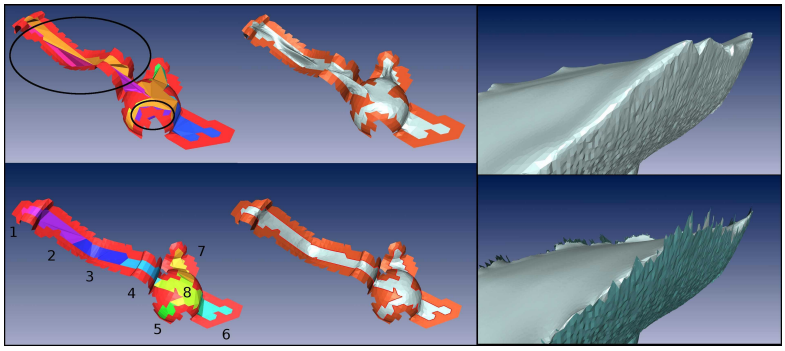

Abbildung 33: Linkes Bild: Die Unterteilung von Filigree aus Tabelle 3. Oben wurden entsprechend Zeile drei Winkelgewichte und eine Schrittweite von drei verwendet. Die bei der Vereinfachung entstandene Triangulierung war so ungünstig, das für die Unterteilung kleine Bereiche am Rand abtrennt (kl. Kreis) und den langen Arm in längs halbiert (gr. Kreis). Das daraus resultierende schlechte Ergebnis ist oben rechts zu sehen. Für die Unterteilung unten links wurden Flächengewichte verwendet, welche sich für die ebene Begrenzung hier besser eignen. Entsprechend Zeile Sechs wurde eine Schrittweite von 5 gewählt. Es entsteht eine sinnvolle Unterteilung in acht Teillöcher. Das Endergebnis entspricht dem Fall ohne Beschleunigung. Rechtes Bild: Bei dem Modell Large ergeben sich die Qualitätseinbusen aus der schlechteren Randanbindung. Oben ohne Beschleunigung mit Winkelgewichten, unten mit Beschleunigung (siehe letzte Zeile der Tabelle).

Die in dieser Arbeit vorgestellte Erweiterung mit dem größten Nutzen ist wahrscheinlich die Beschleunigung durch Unterteilung großer Löcher. Erst mit ihr wird es praktikabel große Löcher zu füllen. In Tabelle 3 werden anhand einiger Beispiele die Möglichkeiten und Grenzen der aktuellen Implementierung gezeigt.Die Grenzen zeigen sich einmal durch eine höhere Wahrscheinlichkeit für Überschneidungen, andererseits in einer teilweise stark von der Originalfüllung abweichenden Oberfläche. Mit Original ist die Füllung ohne Beschleunigung gemeint. Während der Versuche mit dem Algorithmus hat sich gezeigt, dass starke Abweichungen vor allem mit Winkelgewichten vorkommen. Wenn die Randgeometrie langgezogen und das Loch dünn ist können Flächengewichte verwendet werden, mit ihnen entstehen geringere Abweichungen.

Speziell für die bei der Beschleunigung häufiger auftretenden Überschneidungen ist eine weitere hier vorgestellte Erweiterung sinnvoll. Mit ihr wird es möglich, viele der Überschneidungen in einer Nachbearbeitung zu entfernen. Oft können die Fehldreiecke entfernt werden, ohne dass sich das Aussehen der Oberfläche ändert, in einigen Fällen entstehen aber im Verhältnis zur Umgebung sehr große Dreiecke. Diese fallen einmal in der Wireframe-Darstellung auf, andererseits bilden sie so im Modell ebene Regionen in einer eventuell gekrümmten Umgebung.

Ob die Beschleunigung in einem konkreten Fall anwendbar ist, hängt von der Geometrie des Loches und von den verfolgten Zielen ab. Wenn

das Loch die Verwendung von Flächengewichten zulässt und bei der Lochreparatur nur die vollständige Verschließung des Loches im Vordergrund steht, spielen Überschneidungen eventuell keine Rolle. Interessant ist in Tabelle 3, dass eine größere Beschleunigung auch zu besseren Ergebnissen führen kann, denn durch Änderungen der Parameter entsteht eine andere Unterteilung.

Keine in dieser Arbeit entwickelte Verbesserung, aber eine interessante Möglichkeit für den Nutzer ist, wie schon mehrfach angesprochen, die Wahl bei den Gewichten zwischen Winkel und Fläche. Der Standard ist das Winkelgewicht. Flächengewichte bieten in Abhängigkeit vom jeweiligen Loch eine höhere Qualität, allgemein wird die Ausführung mit ihnen schneller, da keine Winkel mehr berechnet werden müssen.

Alle drei der in diesem Abschnitt besprochenen Optionen haben den selben Nachteil: Es gibt keine Garantie, wie das Ergebnis ausfällt. Für die Beschleunigung kann garantiert werden, dass die Berechnung mit jeder Erhöhung des Schiebereglers schneller abläuft. Wie sich dies aber auf die Qualität auswirkt, kann vorher nicht beurteilt werden; man muss es ausprobieren. Das gleiche gilt für eine Anpassung der Gewichte. Mit der Wahl von Flächengewichten verläuft die Berechnung schneller. Mit ein wenig Erfahrung erkennt man meist vorher, wann mehr auf die Fläche geachtet werden sollte, die Software kann dies aber momentan nicht automatisch entscheiden.

Bei Aktivierung der Entfernung von Überschneidungen erhält man nach dessen Ausführung eine Aussage, ob es noch lokale Überschneidungen gibt. Gab es vorher keine, wird es auch danach keine geben und das Verfahren kommt schnell zum Ende. Wenn es Überschneidungen gibt, dann konnte das Verfahren in allen hier durchgeführten Versuchen die Anzahl der Probleme verringern oder Überschneidungen ganz entfernen. Teilweise sind aber durch das Entfernen vieler Dreiecke an einem Ort verhältnismäßig große, auffällige Dreiecke entstanden.

Eine letzte umgesetzte Verbesserung betrifft die Erweiterung der initialen Triangulierung. Mit ihr wurde die Anzahl getesteter Dreieckskombinationen verdoppelt, wodurch eine Verbesserung der eingefügten Oberfläche erreicht wird. Der Nachteil ist die nun auch verdoppelte Laufzeit.

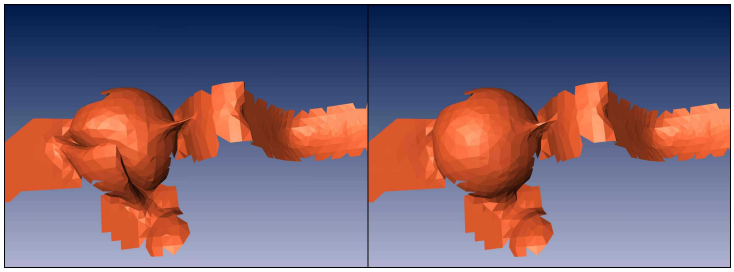

Abbildung 34: Links ist die herkömmliche Triangulierung zu sehen, rechts die erweiterte Version mit allen Möglichkeiten. Die Faltung auf der Kugel ist beseitigt.

Der Algorithmus von Liepa wurde bis hierher vollständig implementiert und er selbst sowie alle daran vorgenommenen Erweiterungen wurden besprochen. Nun soll er mit acht vorhandenen Softwarelösungen verglichen werden:

- dem kommerziellen **Rapid Form 2004**, es ist auf den Umgang mit Daten von 3D Scannern angepasst
- dem kommerziellen **Alias Surface 2010**, dient der 3D-Flächenmodellierung und zielt auf Produkt- und Automobildesign
- dem kommerziellen **Rhino 4.0**, es richtet sich an Designer und deckt den kompletten Prozess von der Skizze bis zum Rendering ab
- dem frei verfügbaren **MeshLab 1.2.1**, es ist speziell an die Verarbeitung von Dreiecksnetzen angepasst
- **JavaView 3.95.001** von Konrad Polthier (www.javaview.de), ein 3D Geometriebetrachter, der sich leicht in andere Programme und in Webseiten einbinden lässt
- dem frei verfügbaren **ReMesh 2.0**, ein Programm zur Bearbeitung von Dreiecksnetzen, zum Füllen von Löchern ist der Algorithmus von Liepa implementiert
- **PolyMender 1.7**, der Implementierung zur Arbeit von [Ju04]
- **Volfill 1.0**, der Implementierung zur Arbeit von [DMGL02]

Einer freien Implementierung eines **Advancing Front** Algorithmus von Schreiner et al. ist im Internet verfügbar. Leider lieferte das Programm auch nach mehreren Stunden kein Ergebnis.

Bedienbarkeit im Vergleich

Als Hauptmerkmal für die Bedienbarkeit wird die Anzahl der benötigten Nutzeraktionen gezählt, die nach dem Laden der Oberfläche nötig sind, um alle Löcher zu reparieren. Von Alias abgesehen verwenden alle Programme gängige Menüs und Fenster, die dem Durchschnittsnutzer bekannt sein sollten und somit kein besonderes Hindernis darstellen. Alias weicht etwas vom der normalen Darstellung ab, was aber nach einer kurzen Gewöhnung kein Problem mehr darstellt. Die Anzahl der Aktionen liefert so eine direkte Aussage über die Dauer und den Aufwand der nötig ist, um alle Löcher einer Oberfläche zu reparieren. Da die Benutzungsoberfläche auf Standardhardware keine Verzögerungen bei der Bedienung aufweist, hängt die letztendlich benötigte Zeit vom Nutzer und seinen Gewohnheiten ab und soll hier wegen der geringen Aussagekraft nicht untersucht werden. In der Tabelle werden zusätzlich grundlegende Eingriffsmöglichkeiten angegeben.

	Rapidform	Alias Surf.	Rhino	MeshLab	JavaView	ReMesh
Bedienbarkeit	**5**	**4 (3*)**	**4**	**4**	**3**	**2**
Einflussmögl.	Flat, Smooth oder Curvature	max. Lochgröße, Fair, Taut		max. Lochgröße		max. Lochgröße, Refine und Smooth
Aufbau	Menü, Einstellungsfenster					

	PolyMend.	Volfill	EarCut	Ruled	MinSurf.	Liepa
	1[2]	**3[3]**	**6**	**6**	**6**	**6**
	Genauigkeit	sehr viele Parameter		Start- und Endpunkt		Refine und Smooth, Beschleunigung
	Konsole		Menü, Einstellungsfenster			

* Nach dem Programmstart muss die Funktion ausgewählt werden aus einem Menü, danach liegt sie oben auf

[2] Ein Konsolenbefehl ist nötig

[3] Mittels drei Konsolenbefehlen muss ein normales Mesh in das VRI-Format konvertiert werden, dann repariert und am Ende rückkonvertiert werden

Tabelle 4: Bedienbarkeit im Vergleich

Wie in Tabelle 4 angegeben nutzen bis auf die beiden Konsolenanwendungen alle Programme gängige grafische Bedienelemente. Die Bedienung bereitete bei keiner Anwendung Probleme. Die Algorithmen in

den letzten vier Spalten sind aus ZIBAmira®. Verglichen mit den anderen Programmen benötigt man hier ein paar Mausklicks mehr um Löcher zu füllen. Dabei muss aber wie bei den meisten hier gezeigten Programmen beachtet werden, dass das Füllen von Löchern nicht die einzige Aufgabe ist und die Anzahl der nötigen Nutzeraktionen auch vom sonstigen Funktionsumfang abhängt.

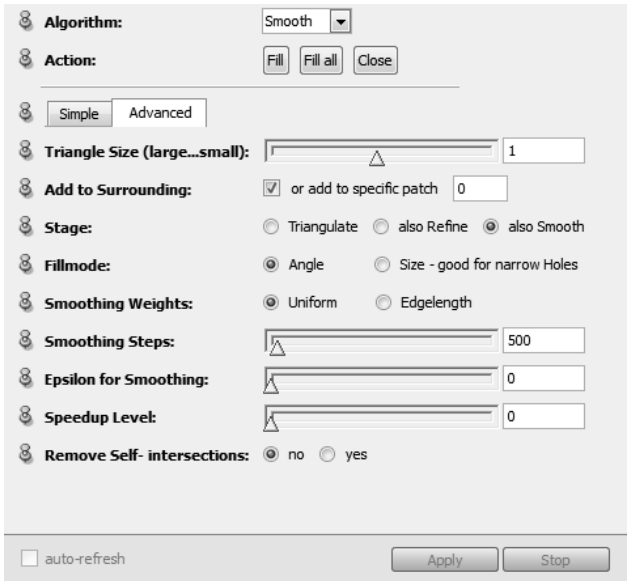

Abbildung 35: Zu sehen sind die erweiterten Einstellungen des Liepaalgorithmus

Für den in dieser Arbeit implementierten Algorithmus von Liepa wurden die möglichen Einstellungen in einem Fenster mit zwei Tabs zugänglich gemacht. Auf dem „Advanced-Tab“, welcher in Abb. 35 zu sehen ist, werden alle Einstellungen angezeigt. Im Bild sind die Standardeinstellungen zu sehen. Auf einem vereinfachten Tab namens „Simple“ werden nur die obersten zwei Einstellmöglichkeiten angezeigt. Der Algorithmus liefert auch mit den Standardeinstellungen gute Ergebnisse, um den Anwender nicht mit zu vielen Details zu überfordern wird initial nur die vereinfachte Version auf dem „Simple-Tab“ angezeigt. Hier sollen kurz alle Optionen der Reihe nach wie in Abb. 35 zu sehen erklärt werden:

- die erste Option erlaubt die Wahl des Verfahrens, momentan ist der Algorithmus von Liepa ausgewählt, welcher in Amira „Smooth“ heißt
- in der nächsten Zeile sind drei Knöpfe zu sehen, beginnend von links kann damit das aktuelle Loch oder alle Löcher verfüllt werden, oder die Einstellungen können geschlossen werden

- die beiden Tabs, momentan ist der „Advanced-Tab“ ausgewählt, welcher alle Einstellungen enthält
- ein Schieberegler, welcher die Anzahl der bei der Verfeinerung eingefügten Punkte und somit auch die Dreiecksgröße bestimmt, von links nach rechts werden immer mehr Punkte eingefügt und die entstehenden Dreiecke werden kleiner
- wenn in der nächsten Zeile im ersten Feld ein Haken gesetzt ist, werden die neuen Dreiecke zum am Loch vorherrschenden Material hinzugefügt, alternativ kann der Nutzer das Material in dem Eingabefeld durch seinen Index festlegen
- nun kann festgelegt werden, ob der Algorithmus nur das Loch triangulieren soll, oder ob er zusätzlich noch neue Punkte einfügen und diese glätten soll
- mit „Fillmode“ können die Gewichte der initialen Triangulierung zwischen Winkel und Fläche umgeschaltet werden, meist bringt die Einstellung Winkel die besten Einstellungen, aber besonders bei langen, schmalen Löchern kann eine Bevorzugung einer minimalen Fläche von Vorteil sein
- die Anpassung der Gewichte der Glättung, Uniform läuft dank der geringeren Dämpfung schneller ab, die Gewichtung durch die Kantenlänge hat in Versuchen nur bei stark variierenden Kantenlängen zu einem besseren Ergebnis geführt
- die Anzahl der Glättungsschritte, je mehr Punkte bei der Verfeinerung entstanden sind, desto länger dauert die Glättung und desto höher sollte dieser Wert gewählt werden, da damit aber auch die Laufzeit steigt wurde er nicht automatisch angepasst
- mit dem Epsilon kann die Glättung schon vor Erreichen der maximalen Schrittanzahl beendet werden, wenn die durchschnittliche Änderung der Punkte zu gering wird
- mit dem Regler der Beschleunigung legt man die Geschwindigkeit der initialen Triangulierung fest, je größer der Wert, desto schneller geht es, bei Null ist die Beschleunigung deaktiviert, ein höherer Wert kann auch zu einem besseren Ergebnis führen

- mit der letzten Option kann man bestimmen, ob Probleme mit sich schneidenden Dreiecken behoben werden sollen, was aber nicht immer funktioniert und das Ergebnis zumindest optisch verschlechtern kann

- ganz unten rechts im Bild ist ein ausgegrauter „Stopknopf“ zu sehen, er wird aktiv wenn der Algorithmus läuft, mit ihm kann man den Algorithmus abbrechen, beispielsweise wenn er unerwartet langsam läuft, falls die initiale Triangulation zu diesem Zeitpunkt bereits abgeschlossen war, werden die aktuell vorhandenen Dreiecke trotzdem eingefügt, man kann so bei der Glättung eine sehr hohe Schrittanzahl einstellen und dann einfach abbrechen, wenn man keine Zeit mehr hat

Geschwindigkeit und Plausibilität

Jedes Programm wird mit fünf verschiedenen Modellen getestet, Tabelle 5. Die Modelle enthalten Löcher, deren Größe zwischen drei und 988 Randpunkten variiert. Die Modelle Fish und Filigree stammen von www.-aimatshape.net, sie enthalten manuell erzeugte Löcher. Face1 und Face2 sind Datensätze eines Laserscanners, Large enthält als Loch die Originalbegrenzung des äußeren Randes des Gesichts aus Face1, in Face1 und Face2 wurden diese Löcher vor dem Test verschlossen.

Alle Programme boten die Möglichkeit, alle Löcher auf einmal zu reparieren, die dazu benötigte Zeit wurde gemessen. Sofern das Programm die dazu benötigte Zeit nicht anzeigte, wurde per Hand gestoppt. Bot ein Programm mehrere Verfahren an, so wurde jedes getestet. Die Ergebnisse wurden abgespeichert und auf Überschneidungen und verbleibende Löcher ausgewertet. Für das Modell Fish wurde die maximale Entfernung der eingefügten Geometrie zum Original, sowie die Volumendifferenz in Amira® ermittelt. Die Werte sind relativ zur Größe der BoundingBox, innerhalb des Modells aber gleich und so vergleichbar. Wichtigstes Vergleichskriterium ist entsprechend dem Fokus dieser Arbeit die Plausibilität, welche in der Tabelle mit „Qualität“ bezeichnet ist. Face2 und Large sind hier mit in Face1 zusammengefasst. Die Bestimmung der Qualität erfolgt im Anhang (Kap. 7) anhand der Ergebnisbilder.

		Rapidform	Alias Surf.	Rhino	MeshLab	JavaView	ReMesh	PolyMender	Volfill	EarCut	Ruled[2]	MinSurf.[2]	Liepa[3]
Fish	*Zeit*	**33/ 1/ 2/ 6**	**6/ 5**	**1**	**0,4**	**2**	**2**	**3,7**	**6,5**	**1**	**1/ -**	**1**	**26,2/ 3,2**
5 Löcher	*Qualität*	**O/ O/ O/ O**	**+/ O**	**O**	**+**	**O**	**+**	**O**	**-**	**O**	**-/ -**	**O**	**+/ O**
	max Entfernung	23/ 31/ 31/ 31	28/ 30	30	16	30	22	30	31	28	26/ 30	27	19/ 29
	Volumendiff.	25/ 14/ 17/ 14	11/ 14	14	13	15	12	16	15	15	16/ 15	16	12/ 15
	Restlöcher	7/ 2/ 1/ 1	0/ 0	6	0	0	0	0	9	0	0/ 169	383	0/ 0
	Überschn.	0/ 0/ 2/ 0	0/ 0	0	0	58	2	0	0	429	402/ 289	613	0/ 0
Filigree	*Zeit*	**42/ 1/ 2/ 3**	**4/ 3**	**2**	**0,3**	**7**	**2**	**10**	**3,2**	**1**	**1/ -**	**1**	**14/ 1,2**
3 Löcher	*Qualität*	**-/ -/ -/ -**	**+/ +**	**-**	**O**	**O**	**+**	**-**	**-**	**-**	**-/ -**	**-**	**+/ +**
	Restlöcher	41/ 3/ 2/ 2	0/ 0	13	0	0	0	0	9	0	0/ 104	217	0/ 0
	Überschn.	2/ 0/ 0/ 72	12/ 11	86	12	38	2	0	0	251	428/ 300	1670	2/33
Face1	*Zeit*	**56/ 2/ 3/ 7**	**4/ 3**	**3**	**0,3**	**7**	**2**	**7,6**	**7,1**	**3**	**3**	**4**	**21,1/ 10,6**
26 Löcher	*Qualität**	**O/ -/ -/ -**	**+/ +**	**-**	**+**	**O**	**+**	**O**	**-**	**O**	**-**	**O**	**+/ +**
	Restlöcher	102/ 6/ 7/ 4	0/ 0	12	0	0	0	0	36	1	0	376	0/ 0
	Überschn.	0/ 13/ 35/ 0	5/ 8	13	102	176	41	0	0	217	409	543	30/ 40
Face2	*Zeit*	**95/ 3/ 5/ 7**	**3/ 3**	**3**	**0,2**	**2**	**1**	**8,6**	**6,5**	**7**	**7**	**8**	**25,6/ 11,8**
80 Löcher	*Restlöcher*	564/ 5/ 5/ 5	0/ 0	12	1	3	1	0	2	3'	1'	528'	5'/ 3'
	Überschn.	0/ 25/ 28/ 33	114/ 117	18	86	159	254	0	0	204	271	1057	98/ 50
Large	*Zeit*	**1023/ 6/ 6/ 7**	**268/ 207**	**1**	**1,1**	**6**	**5**	**6**	**4,9**	**1**	**1**	**14**	**547/ 70,8**
1 Loch	*Restlöcher*	2/ 1/ 2/ 2	0/ 0	0	0	0	0	0	5	0	0	562	0/ 0
	Überschn.	0/ 0/ 0/ 4	1219/ 957	2225	0	1086	78	0	0	798	172	620	24/ 239

Rechnerkonfiguration: Rapidform(Intel Xeon 3,2Ghz mit 3,25GB Ram), Volfill(Intel Atom 1,6Ghz mit 1 GB Ram, alle Anderen(Intel Core2 1,66Ghz mit 2 GB Ram)
Einstellungen: soweit nicht anders angegeben auf Default, PolyMender(Octreetiefe 9), Rhino(Konnektivität musste in MeshLab hergestellt werden)
Aufbau: Abgesehen von MeshLab, ReMesh, Volfill und Liepa sind die Zeiten handgestoppt, Überschneidungen wurden in MeshLab gezählt, Überschneidungen in Amira

∗ Gilt auch für Face2 und Large

' Lochtracing hatte bei Face2 Probleme, der Fehler konnte erst nach Beendigung dieser Arbeit behoben werden

[2] bei Ruled Self und Min entstehen Löcher, weil Punkte auf Kanten gelegt werden und die Topologie so nicht geschlossen ist

[3] Die Einstellungen für die optimierte Messung: Fish(nur 50 Smoothingschritte, Lochreparatur, Beschleunigungsstufe 6), Filigree(nur 50 Smoothingschritte, Beschleunigungsstufe 6), Face1(nur 50 Smoothingschritte, min. Oberfläche, Beschleunigungsstufe 1), Face2(nur 50 Smoothingschritte, min. Oberfläche, Beschleunigungsstufe 3), Large (Beschleunigungsstufe 2)

Tabelle 5: Die verschiedenen Programme im Vergleich, Laufzeiten sind in Sekunden angegeben.

5.3 Diskussion

Zur Zusammenfassung der Verbesserungen. Mit ihnen besteht die Möglichkeit Löcher schneller zu verschließen, wobei aber mit einer schlechteren Oberfläche und mehr Überschneidungen gerechnet werden muss. In Kombination damit oder auch eigenständig bietet eine weitere Erweiterung die Option, Überschneidungen zu entfernen. Erfahrene Nutzer können durch Anpassung der Gewichte die Triangulation beeinflussen und allgemein wurde die Triangulation unter einem gewissen Mehraufwand hinsichtlich ihrer Qualität verbessert.

In ZIBAmira® benötigt man ein paar Mausklicks mehr, bis man Löcher einer Oberfläche reparieren kann, der geringe Unterschied wird nicht als gravierend angesehen. Die Benutungsoberfläche bietet einen einfachen Modus und einen erweiterten Modus. Auch mit den Standardeinstellungen können Löcher gut verschlossen werden. Die Oberfläche wird gegenüber den anderen Programmen als gleichwertig angesehen.

Die Laufzeit der Standardimplementierung ist die zweitschlechteste im Testfeld, nur die Volumenmethode von Rapidform ist langsamer. Die als Erweiterung umgesetzte Beschleunigung verbessert die Situation. Allerdings muss man dabei teilweise probieren, welche Einstellungen auch ein gutes Ergebnis liefert. Eine Verbesserung der Beschleunigung wie im entsprechenden Kapitel dieser Arbeit vorgeschlagen ist nötig, um die Funktion sinnvoll nutzbar zu machen.

Die Qualität der erzeugten Oberflächen und die dabei eingeführten Überschneidungen sind mit den guten Ergebnissen von Alias Surface und ReMesh vergleichbar. Bei ReMesh fiel die stärkere Glättung des Endergebnisses auf. In der hier gezeigten Implementierung wären für ähnliche Ergebnisse weit mehr Iterationen nötig, was zu einer stark erhöhten Laufzeit führt. Somit sind genauere Vergleiche mit ReMesh und dessen Implementierung sinnvoll.

6 Fazit und Ausblick

In der ersten Hälfte dieser Arbeit wurden verschiedene Algorithmen zur Reparatur von Löchern vorgestellt. Einige von ihnen wurden im Detail untersucht, um eine geeignetes Verfahren zur Implementierung zu finden. Die anderen wurden überblicksartig behandelt und die angewandten Wahrnehmungsprinzipien herausgearbeitet.

Im zweiten Teil wurde das gefundene Verfahren von Liepa implementiert und einige Verbesserungen wurden erarbeitet und teilweise umgesetzt. Mit dieser Softwarelösung ist es möglich, Löcher wie sie häufig bei Modellen von Laserscannern entstehen zu reparieren. Die erzeugten Oberflächen sind hinsichtlich ihrer Qualität mit ähnlichen gängigen Produkten vergleichbar.

Der Vergleich diesen Programmen hat aber auch Schwächen in der Laufzeit aufgezeigt. Die Software ReMesh verwendet den selben Algorithmus, ist aber in ihrer Ausführung schneller. Ziel weiterer Arbeiten sollte es sein, durch einen genaueren Vergleich mit ReMesh Unterschiede in der Implementierung herauszuarbeiten und gegebenenfalls die hier vorgenommene Umsetzung zu verbessern oder die Datenstrukturen von ZIBAmira® zu optimieren.

Zur Verbesserung der Laufzeit wurde in dieser Arbeit bereits eine Erweiterung des Algorithmus vorgenommen, welche die Ausführung deutlich verkürzt, aber auch die Qualität negativ beeinflusst. Vorschläge für weitere Anpassungen dahingehen wurden gemacht.

Ein offener Punkt dieser Arbeit ist, wie die Benutzungsoberfläche von den Anwendern angenommen wird. Momentan sind viele Optionen enthalten, die nur selten benötigt werden. Diese können eventuell entfernt werden. Durch eine automatische Anpassung einiger Werte könnte die Bedienung ebenfalls vereinfacht werden.

7 Anhang

Für alle in Tabelle 5 aufgeführten Modelle sind hier Ergebnissbilder der Programme zu sehen, aus ihnen leidet sich die Beurteilung der Plausibilität ab. Für die ersten beiden Modelle wurde manuell Löcher erzeugt. Da hier die Originalgeometrie vorhanden war konnten Differenzbilder erzeugt werden. Die Differenz zwischen Original und dem jeweiligen Ergebnis wird dabei farblich auf dem Originalmodell wiedergegeben. Die Beurteilung wird in den Bildunterschriften genauer dargelegt. Da es schwierig ist, Plausibilität in Zahlen zu fassen, spiegelt die Beurteilung nur die Meinung des Autors wider. Durch die nachfolgenden Bilder soll jeder Leser in die Lage versetzt werden, diese Beurteilung nachzuvollziehen und sich selbst eine Meinung zu bilden.

Die Bewertung für jedes Programm setzt sich aus folgenden Punkten zusammen:

- Sind die Löcher sinnvoll und plausibel verschlossen? Da scheinbar keines der Programme Details aus der Umgebung übernimmt, etwa die Wellen auf der Flosse des Fisches, wird dies hier nicht für jedes Bild angemerkt.
- Für die Modelle Fish und Filigree wird der Abstand zur Originaloberfläche anhand der Farbbilder relativ zu den anderen Ergebnissen ausgewertet
- sind störende Artefakte in der neuen Geometrie vorhanden, hebt sie sich stark von der Umgebung ab
- wie ist die Anbindung an die Umgebung

Zur Beurteilung dieser Punkte werden die verschlossenen Löcher betrachtet, noch offene werden erst bei der Gesamtbewertung abgewertet. Bewertet wird mit +, ○, -. Ein Plus steht für eine verhältnismäßig gutes Ergebnis, ein Minus entsprechend für eine niedrige. Die vier beziehungsweise drei Punkte werden zu einem Gesamtwert zusammengefasst, welcher in Tabelle 5 in der Zeile Qualität zu finden ist.

7.1 Modell Fish

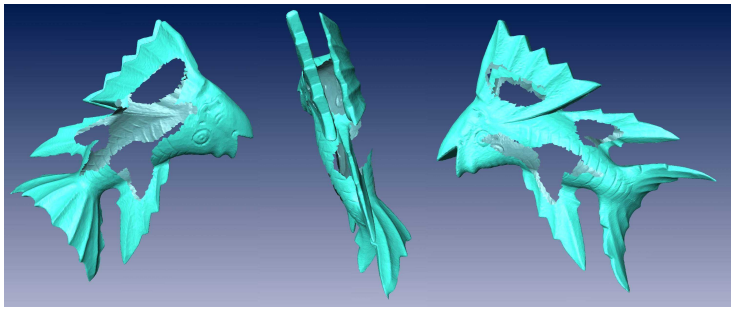

Abbildung 36: Das Eingabemodell mit Löchern.

Abbildung 37: Die Farbskalen für die Bilder dieses Abschnitts. Die linke Skala gilt je für die beiden linken Bilder, die rechte je für die beiden mittleren Bilder. Die Angaben sind Zentimeter. Zum Vergleich, der Fisch hat von Kopf bis Schwanzflosse eine Länge von etwa 40 cm.

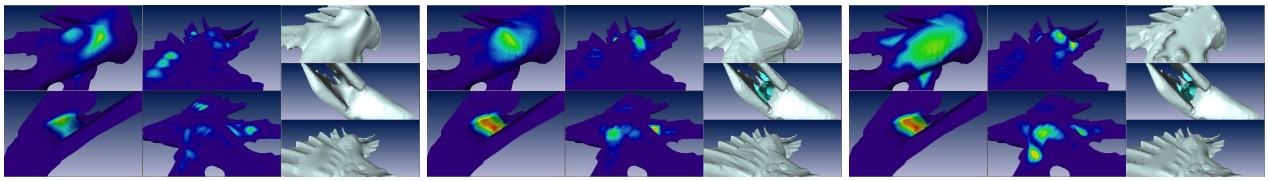

Abbildung 38: Rapidform Volume: ○(offenes Loch am Kopf), ○(nur wenig Grün), +, +, Gesamt: ○ **Rapidform Flat:** ○(offenes Loch am Kopf), ○(nur wenig Rot), ○(besonders im großen Loch kantig), +(trotz fehlender Glättung gut), Gesamt: ○ **Rapidform Smooth:** ○(offenes Loch am Kopf), ○(nur wenig Rot), +, ○(Probleme beim großen Loch), Gesamt: ○

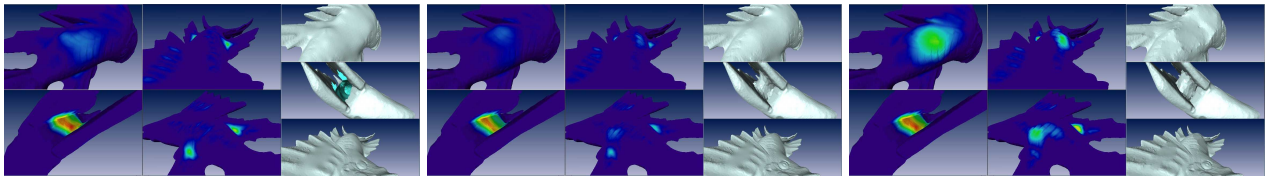

Abbildung 39: Rapidform Curvature: ○(offener „Trichter“ über einem Loch), +(nur wenig Grün), +, +, Gesamt: ○(wegen dem offenen Loch) **Alias Fair:** +, ○(Problem zwischen den Kopfflossen), +, +, Gesamt: + **Alias Taut:** ○(wirkt auf einer Seite eingedrückt), ○(nur wenig Rot), +, +, Gesamt: ○

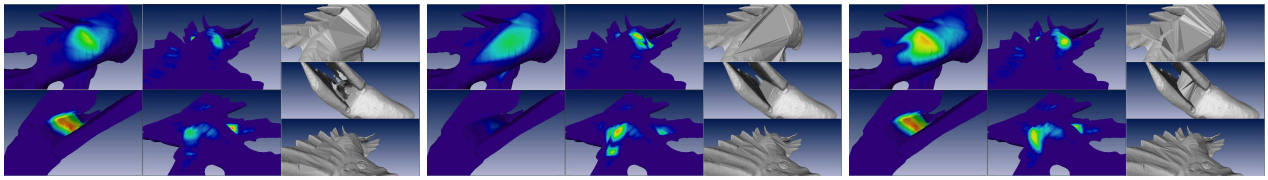

Abbildung 40: Rhino: ○(offenes Loch am Kopf), ○(nur wenig Rot), ○, +, Gesamt: ○ **Meshlab:** +, ○, ○, +, Gesamt: +(Bereich am Kopf ist sehr gut) **Javaview:** ○(großes Loch eingedrückt, Spitze regt heraus), ○(nur wenig Rot), ○, ○, Gesamt: ○

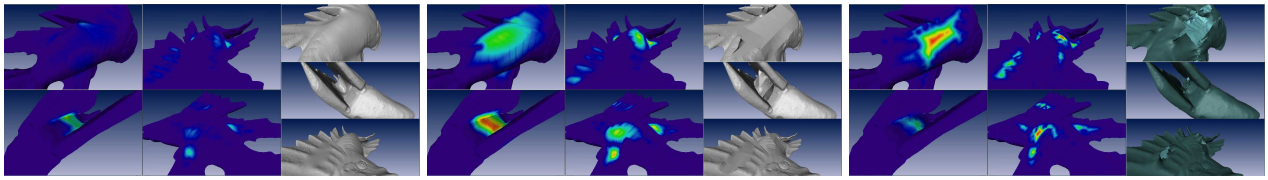

Abbildung 41: Remesh: +, +, +, +, Gesamt: + **PolyMender:** +, ○, ○(Struktur des Octree sichtbar), +(trotz der Zacke beim großen Loch), Gesamt: ○ **Volfill:** -(Löcher größtenteils noch offen mit nach außen gebogener Geometrie am Rand), -(Entfernung wegen der vielen Löcher fehlerhaft), Teile der neuen Geometrie sehen gut aus, andere aber sehr schlecht, Gesamt: -

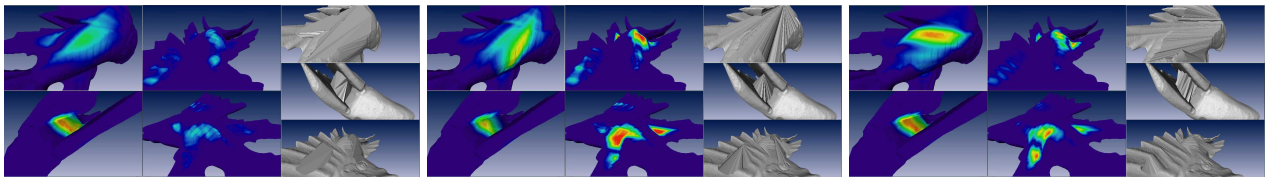

Abbildung 42: Earcut: +, ○(nur wenig Rot), ○, ○, Gesamt: ○ **Ruled Auto:** ○, -, -, ○, Gesamt: - **Ruled Self:** ○(großes Loch problematisch), -, -, ○, Gesamt: -

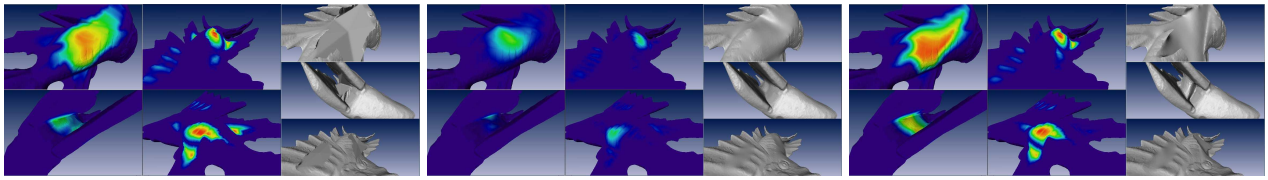

Abbildung 43: Min: +, -, ○, ○, Gesamt: ○ **Liepa Default:** +(Stelle am Kopf sehr gut), +, ○(großes Loch wirkt eingedrückt), +, Gesamt: + **Liepa Speedup:** ○(beide Seiten eingedrückt, Spitze regt heraus), -, ○(großes Loch wirkt wellig), ○, Gesamt: ○

7.2 Modell Filigree

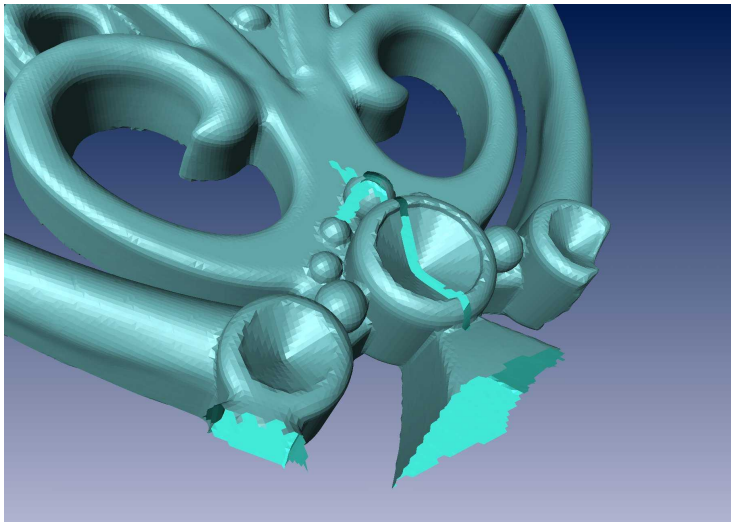

Abbildung 44: Das Eingabemodell mit Löchern.

Abbildung 45: Die Farbskala für die Bilder dieses Abschnitts, die Angaben sind Millimeter. Zum Vergleich, die große Halbkugel, die über das Loch kreuzförmig verläuft, hat einen Radius von 70 mm. Viele Programme hatten im Ergebnis eine weit größere Abweichung als 20 mm, zu Gunsten der besseren Vergleichbarkeit wurden deren Ergebnisse abgeschnitten und mit auf den Maximalwert abgebildet.

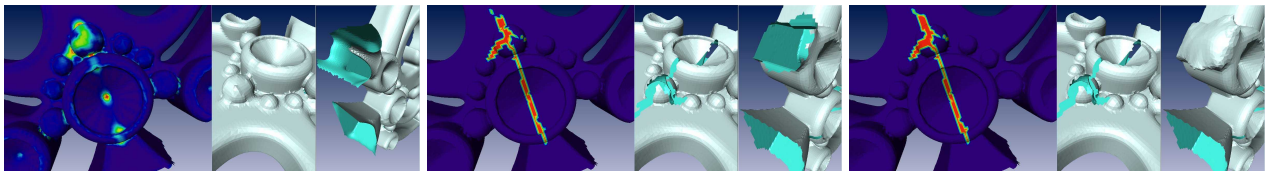

Abbildung 46: Rapidform Volume: +(für das eine verschlossene Loch), ○, +, ○, Gesamt: -(2 von 3 Löchern sind offen) **Rapidform Flat:** kein geschlossenes Loch, Gesamt: - **Rapidform Smooth:** +, (keine Abweichung berechnet), +, +, Gesamt: -(2 von 3 Löchern sind offen)

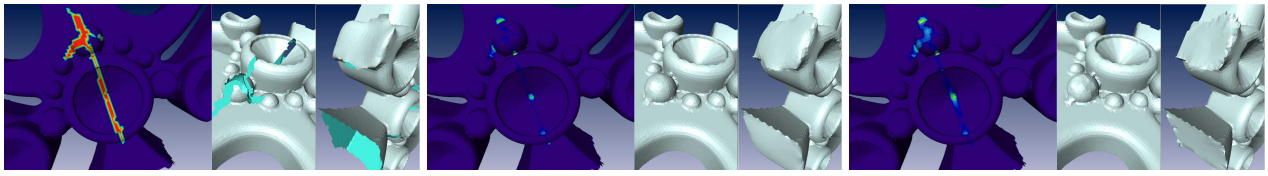

Abbildung 47: Rapidform Curvature: +, (nicht vorhanden), +, ○, Gesamt: - **Alias Fiar:** +, +, +, +, Gesamt: + **Alias Taut:** +, +, ○(Kugel etwas kantig), +, Gesamt: +

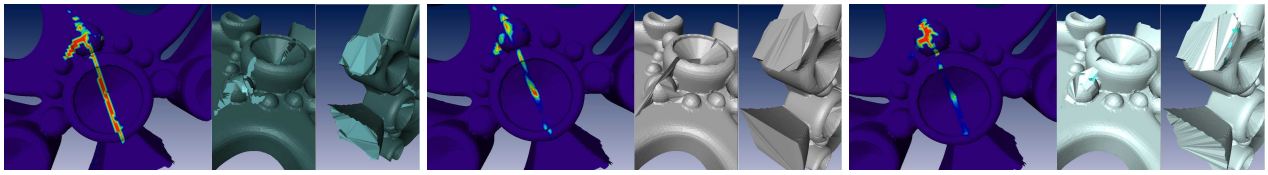

Abbildung 48: Rhino: -(Loch über der Kugel größtenteils offen, Orientierung der Dreiecke in den anderen beiden Löchern variiert), (nicht vorhanden), -, ○, Gesamt: - **Meshlab:** -(Kugel kaum erkennbar), ○, ○, ○, Gesamt: ○ **Javaview:** ○(Kugel etwas besser als Meshlab), ○, ○, ○, Gesamt: ○

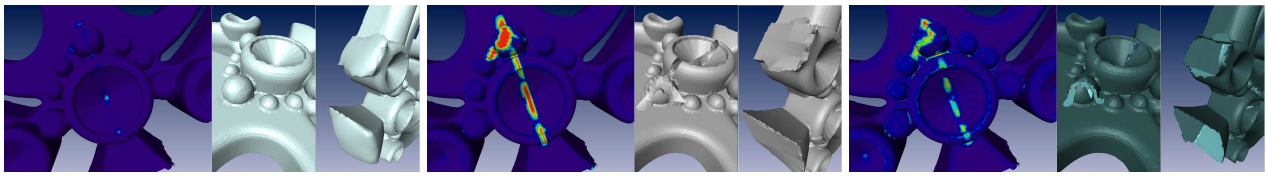

Abbildung 49: ReMesh: +, +, +, +, Gesamt: + **PolyMender:** -(Kugel kaum erkennbar), -, ○, ○, Gesamt: - **Volfill:** -,(kein Loch wirklich verschlossen), Gesamt: -

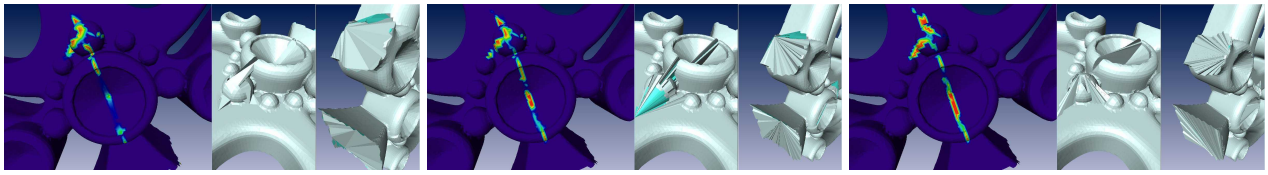

Abbildung 50: Earcut: -(Kugel kaum erkennbar), -, -, -, Gesamt: - **Ruled Auto:** -, -, -, -, Gesamt: - **Ruled Self:** -, -, -, -, Gesamt: -

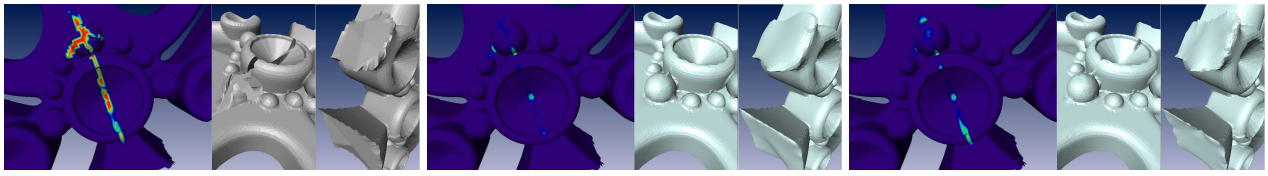

Abbildung 51: Min: -(Kugel kaum erkennbar), -, ○, ○, Gesamt: - **Liepa Default:** +, +, +, +, Gesamt: + **Liepa Speedup:** +, +, +, ○, Gesamt: +

7.3 Modelle Face1, Face2, Large

Die linken beiden Bilder gehören zum Modell Face1, das rechte obere zu Face2 und das rechte untere zum Modell Large. Der Name Large bezieht sich auf die Größe des Loches.

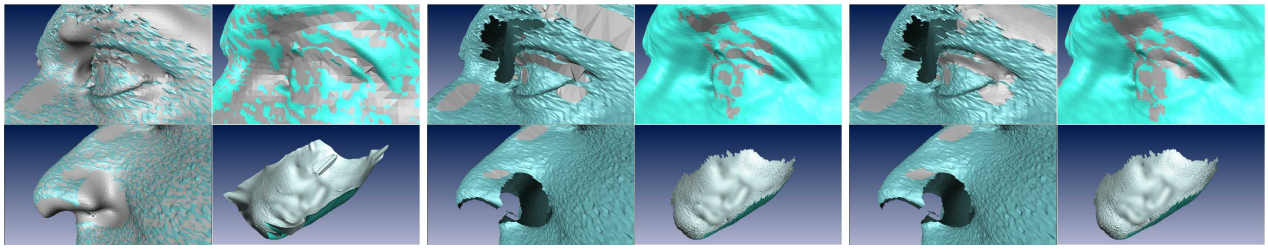

Abbildung 52: Rapidform Volume: ○Large ist problematisch, +, +, Gesamt: ○(ein großes offenes Loch) **Rapidform Flat:** ○, ○, +, Gesamt: -(viele offene Löcher) **Rapidform Smoot:** ○, ○, +, Gesamt: -(viele offene Löcher)

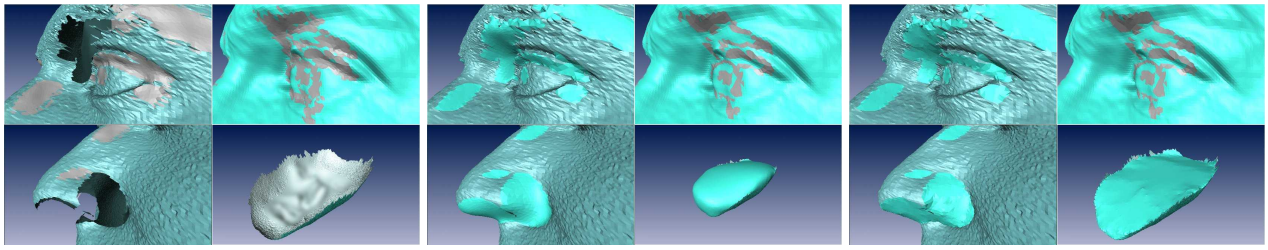

Abbildung 53: Rapidform Curvature: ○, ○, +, Gesamt: -(viele offene Löcher) **Alias Fair:** +, +, ○(überstehende Zacken bei Large), Gesamt: + **Alias Taut:** +, +, ○, Gesamt: +

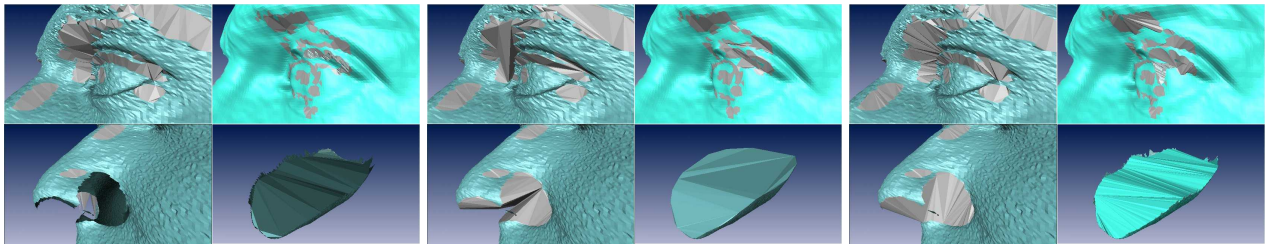

Abbildung 54: Rhino: +, ○, ○, Gesamt: -(Nase noch offen) **Meshlab:** ○(Nase), ○, +, Gesamt: +(Large ist sehr gut verschlossen) **Javaview:** +, ○, ○, Gesamt: ○

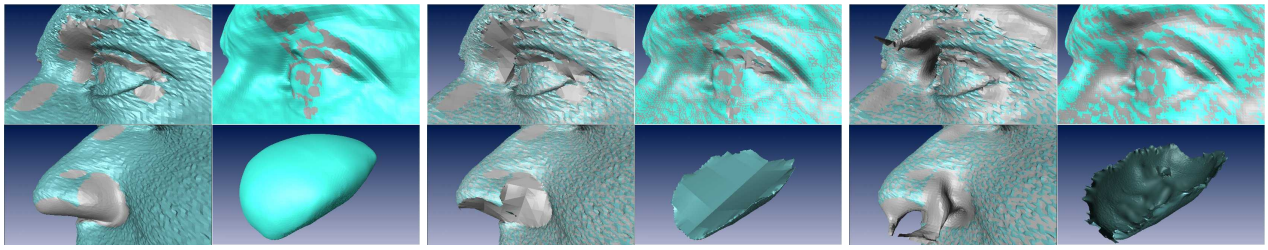

Abbildung 55: ReMesh: +, +, +, Gesamt: + **PolyMender:** ○, ○, +, Gesamt: ○ **Volfill:** -, -, +, Gesamt: -

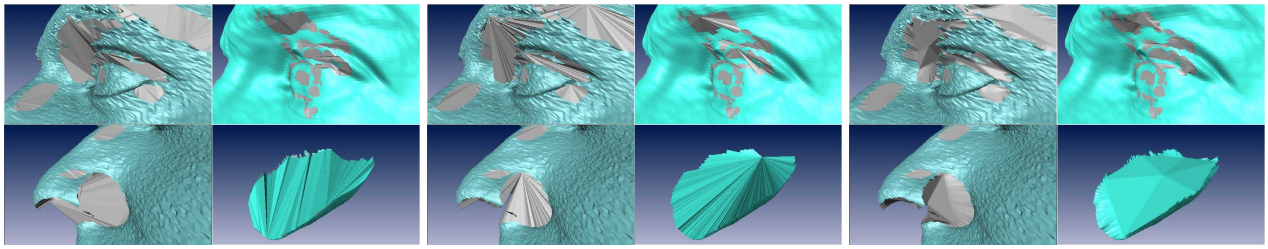

Abbildung 56: Earcut: ○, ○, ○, Gesamt: ○ **Ruled Auto:** ○, -, -, Gesamt: - **Min:** ○, ○, ○, Gesamt: ○

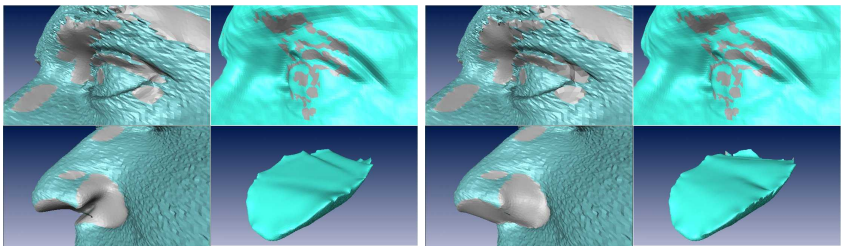

Abbildung 57: Liepa Default: ○, +, +, Gesamt: + **Liepa Speedup:** +, +, +, Gesamt: +

Literatur

[ACP03] Brett Allen, Brian Curless, and Zoran Popović. The space of human body shapes: reconstruction and parameterization from range scans. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 587–594, New York, NY, USA, 2003. ACM.

[ASK+05] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: shape completion and animation of people. *ACM Trans. Graph.*, 24(3):408–416, 2005.

[BB06] Michael Bender and Manfred Brill. *Computergrafik - ein anwendungsorientiertes Lehrbuch*. Hanser Verlag, Muenchen, 2. auflage edition, 2006.

[BF05a] Toby P. Breckon and Robert B. Fisher. Amodal volume completion: 3d visual completion. *Comput. Vis. Image Underst.*, 99(3):499–526, 2005.

[BF05b] Toby P. Breckon and Robert B. Fisher. Non-parametric 3d surface completion. In *3DIM '05: Proceedings of the Fifth International Conference on 3-D Digital Imaging and Modeling*, pages 573–580, Washington, DC, USA, 2005. IEEE Computer Society.

[Bla04] F. Blais. Review of 20 years of range sensor development. *Journal of Electronic Imaging*, 13:231–+, 2004.

[BMVS04] Volker Blanz, Albert Mehl, Thomas Vetter, and Hans-Peter Seidel. A statistical method for robust 3d surface reconstruction from sparse data. In *3DPVT '04: Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium*, pages 293–300, Washington, DC, USA, 2004. IEEE Computer Society.

[BR02] Fausto Bernardini and Holly E. Rushmeier. The 3d model acquisition pipeline. *Comput. Graph. Forum*, 21(2):149–172, 2002.

[BSK05] Gerhard H. Bendels, Ruwen Schnabel, and Reinhard Klein. Detail-preserving surface inpainting. In *The 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST)*, pages 41–48. Eurographics Association, Eurographics Association, November 2005.

[DMGL02] James Davis, Stephen R. Marschner, Matt Garr, and Marc Levoy. Filling holes in complex surfaces using volumetric diffusion. *3D Data Processing Visualization and Transmission, International Symposium on*, 0:428, 2002.

[Ebe06] David Eberly. Triangulation by ear clipping, 2006.

[EET89] Hossam ElGindy, Hazel Everett, and Godfried Toussaint. Slicing an ear in linear time. In *Pattern Recognition Letters*, pages 719–722, 1989.

[Ju04] Tao Ju. Robust repair of polygonal models. *ACM Trans. Graph.*, 23(3):888–895, 2004.

[Ju09] Tao Ju. Fixing geometric errors on polygonal models: A survey. *J. Comput. Sci. Technol.*, 24(1):19–29, 2009.

[KHYS02] Kolja Kähler, Jörg Haber, Hitoshi Yamauchi, and Hans-Peter Seidel. Head shop: generating animated head models with anatomical structure. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 55–63, New York, NY, USA, 2002. ACM.

[KS05] Vladislav Kraevoy and Alla Sheffer. Template-based mesh completion. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, page 13, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[Lie03] Peter Liepa. Filling holes in meshes. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 200–205, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Mei75] G. H. Meisters. Polygons have ears. *The American Mathematical Monthly*, 82(6):648–651, June 1975.

[NYC05] Minh X. Nguyen, Xiaoru Yuan, and Baoquan Chen. Geometry completion and detail generation by texture synthesis. *The Visual Computer (Special Issue for Pacific Graphics2005)*, 21(9–10):669–678, 2005.

[PGSQ06] Seyoun Park, Xiaohu Guo, Hayong Shin, and Hong Qin. Surface completion for shape and appearance. *Vis. Comput.*, 22(3):168–180, 2006.

[PMG+05] Mark Pauly, Niloy J. Mitra, Joachim Giesen, Markus Gross, and Leonidas J. Guibas. Example-based 3d scan completion. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, page 23, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[PMW+08] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. Guibas. Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics*, 27(3):#43, 1–11, 2008.

[PP93] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2:15–36, 1993.

[PR05] Joshua Podolak and Szymon Rusinkiewicz. Atomic volumes for mesh completion. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, page 33, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[SACO04] Andrei Sharf, Marc Alexa, and Daniel Cohen-Or. Context-based surface completion. *ACM Trans. Graph.*, 23(3):878–887, 2004.

[SBSE08] A. Schilling, K. Bidmon, O. Sommer, and T. Ertl. Filling Arbitrary Holes in Finite Element Models. In *Proceedings 17th International Meshing Roundtable*, pages 231–248, 2008.

[Wik] Dreiecksnetz. Online verfügbar unter http://de.wikipedia.org/wiki/Dreiecksnetz; besucht am 7. Juli 2009.

[XGR+06] Songhua Xu, Athinodoros Georghiades, Holly Rushmeier, Julie Dorsey, and Leonard McMillan. Image guided geometry inference. In *3DPVT '06: Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, pages 310–317, Washington, DC, USA, 2006. IEEE Computer Society.