# Retriangulation von Freiformoberflächen nach Polygonreduktion

**Semesterarbeit im Fach Informatik**

vorgelegt von

**David Frank**
Wallisellen, Zürich, CH
Matrikelnummer: 90-912-999

Angefertigt nach PO95 am
Institut für Informatik
der Universität Zürich
Prof. Dr. P. Stucki

Betreuer: PD Dr. Christoph Zollikofer, IFI
Abgabe der Arbeit: 21. August 2002

v1.0

# Zusammenfassung

C. Albrecht hat in seiner Diplomarbeit [Albrecht2000] einen Algorithmus zur Polygonreduktion (Datenreduktion) von triangulierten Freiformoberflächen vorgestellt. Die vorliegende Semesterarbeit erweitert diese Arbeit in zwei Richtungen. Einerseits wird die Originaltreue der datenreduzierten Oberfläche dadurch erhöht, dass Bereiche mit ‘interessante Details’ weniger stark komprimiert werden als andere. Als Kriterium für ‘interessant’ wird die lokale Oberflächenkrümmung verwendet. Andererseits wird die bei Albrecht resultierende Polygonoberfläche wieder in eine triangulierte Form zurückgeführt.

# Inhaltsverzeichnis

# 1. Einleitung

Bei der Untersuchung biologischer Strukturen – beispielsweise mittels Computer-Tomographe – fallen mitunter riesige Datenmengen an, die visualisiert werden müssen. Oft möchte man dabei zugunsten einer einfacheren Handhabung die enorme Datenmenge etwas reduzieren ohne dabei aber wichtige Infomationen zu verlieren. C. Albrecht hat in seiner Diplomarbeit [Albrecht2000] einen entsprechenden Datenreduktions-Algorithmus vorgestellt, der in Analogie zum Nistverhalten von Vögeln (die offenbar beim Nisten versuchen, einen Minimalabstand untereinander einzuhalten) Punkte auf die Objektoberfläche verteilt und sie solange wandern lässt, bis sich ein stabiler Zustand ergibt (bzw. ein Fliessgleichgewicht). Mit dieser Untermenge der originalen Datenpunkte lässt sich eine Abbildung des Originalobjekts erstellen, die zwar weniger Details zeigt aber auch wesentlich einfacher zu handhaben ist.

Die vorliegende Semesterarbeit erweitert diese Arbeit in zwei Richtungen, die gänzlich unabhängig voneinander sind.

Einerseits soll die Originaltreue der datenreduzierten Oberfläche dadurch erhöht werden, dass Bereiche mit ‘interessante Details’ weniger stark komprimiert werden als andere. Die Aufgabenstellung und Motivation folgt in Kapitel 2.1.

Andererseits wird soll die bei Albrecht bei der Datenreduktion resultierende Polygonoberfläche wieder in eine triangulierte Form zurückgeführt werden. Die Aufgabenstellung und Motivation dazu folgt in Kapitel 3.1. auf Seite 8.

# 2. Krümmungsabhängige Verteilung

## 2.1. Aufgabenstellung

Die in der Einleitung beschriebene Polygon-Reduktion wird hauptsächlich unternommen, um das Datenvolumen eines Objekts zu reduzieren und nimmt dabei einen Verlust an Information in Kauf. Sie gleicht darin den bekannten verlustbehafteten Kompressionsverfahren (lossy compression) wie JPEG (für 2D-Bilddaten) oder MP3 (für Audio-Daten). Während verlustfreie Kompressionsverfahren primär an ihrem Kompressionsgrad und an der benötigten Rechenzeit gemessen werden, spielt es bei den verlustbehafteten Kompressionsverfahren eine zentrale Rolle, wie gut sich für den Benutzer aus den verbleibenden Daten die für ihn interessanten Aspekte des Originals rekonstruieren lassen. Ein guter Audio-Kompressions-Algorithmus etwa beschränkt sich nicht darauf, uniform Daten zu reduzieren (downsampling), sondern versucht, zuerst die für den Hörer am wenigsten wichtigen Klanginformationen zu eliminieren (beispielsweise hohe Frequenzen).
Genauso interessieren bei Freiformoberflächen gewisse Objektmerkmale stärker als andere. Am interessantesten sind in der Regel Stellen, bei denen die Oberflächenkrümmung gross ist. Bei der Abbildung eines Kopfes wären wahrscheinlich Details von Nase, Ohren und Kinn interessant, während für Stirn und Hinterkopf eine kleinere Bildauflösung genügen.
Aus diesem Grund soll die Polygon-Reduktion krümmungsabhängig gemacht werden.

## 2.2. Lösungsansatz

Der 'Nesting Birds'-Algorithmus [Albrecht2000] versucht, Punkte gleichmässig über eine Oberfläche zu verteilen indem die Punkte so über die Oberfläche wandern, dass ihre gegenseitigen Abstände maximiert werden. Das gibt uns einen einfachen Ansatzpunkt, um in "interessanten" Regionen mit hoher Krümmung eine höhere Punktdicht (mehr Details) zu erhalten. Wir vergrössern die Abstände in Bereichen hoher Krümmung künstlich und verhindern dadurch, dass dort die Punkte bei der Wanderung auseinanderlaufen. Damit führen wir eigentlich weiterhin eine gleichmässige Verteilung durch – aber in einem gekrümmten Raum.

Die Distanz *d* zwischen zwei Punkten *a* und *b* auf der Objektoberfläche ist:

$$\left|\vec{d}\right| = \left|\vec{b} - \vec{a}\right| = \sqrt{\left(b_x - a_x\right)^2 + \left(b_y - a_y\right)^2 + \left(b_z - a_z\right)^2}$$

**Abbildung 2-1: Distanz zweier Punkte**

Wir benutzen das Skalarprodukt der Normalenvektoren $n_a$ und $n_b$ in den Punkten a und b als Mass für die Krümmung. Für normalisierte Normalenvektoren ist das Skalarprodukt gleich dem Cosinus des Zwischenwinkels $\alpha$. Die Gewichtete Distanz $d_w$ zwischen zwei Punkten *a* und *b* wird somit:

$$\left|\vec{d}_w\right| = \left|\vec{b} - \vec{a}\right| \cdot \left(1 + \frac{w-1}{2} \cdot \left(1 - \vec{n}_a \bullet \vec{n}_b\right)\right)$$

$$= \left|\vec{d}\right| \cdot \left(1 + \frac{w-1}{2} \cdot (1 - \cos\alpha)\right)$$

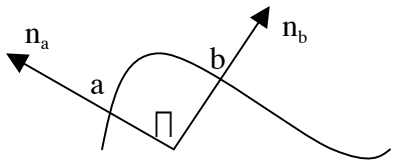



**Abbildung 2-2: Gewichtete, winkelabhängige Distanz zweier Punkte**

Der Gewichtungsfaktor *w* bestimmt die Krümmungsabhängigkeit der Verteilung (*w*=1: kein Einfluss der Krümmung, *w*>1: grosse Krümmung erhöht die Punktdistanz). Das Skalarprodukt der Normalenvektoren wird 0 wenn sie im rechten Winkel (90°) zueinander stehen und <0, wenn sie in einem stumpfen Winkel zueinander sind. Bei einer maximalen Krümmung (Normalen in entgegengesetzter Richtung) wird die Distanz mit dem Faktor *w* multipliziert.

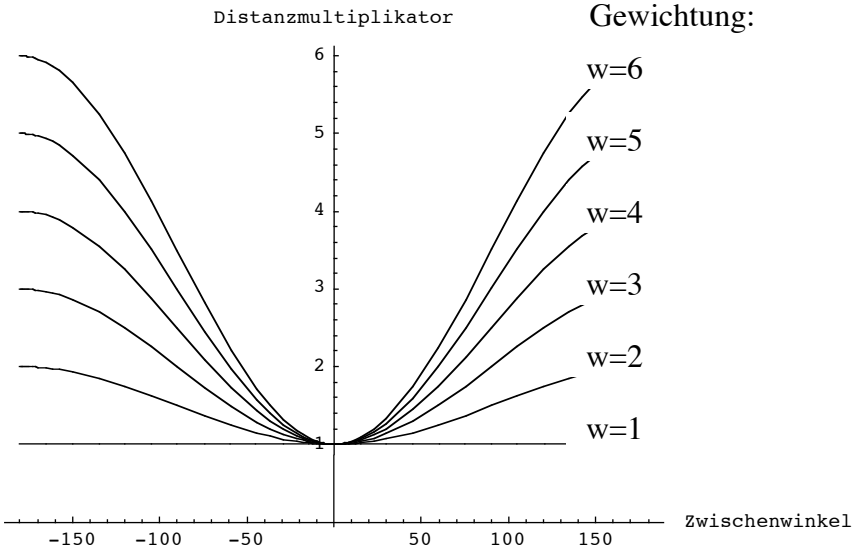
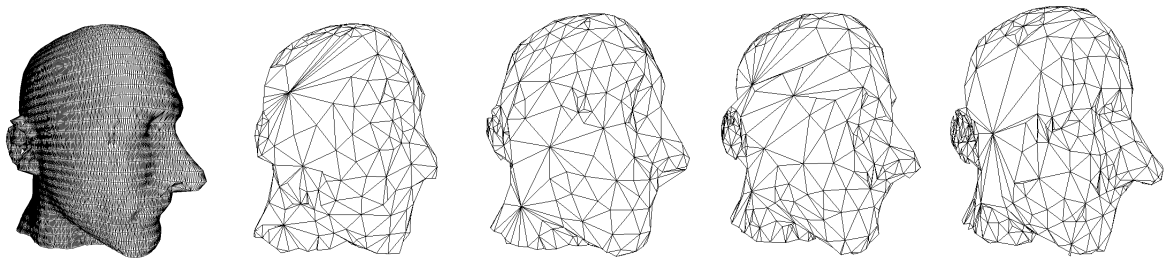



**Abbildung 2-3: Winkelabhängigkeit der Gewichtung**

## 2.3. Resultate

Die folgende Bildreihe zeigt den Effekt der Gewichtung am Beispiel von Computertomographie-Daten eines Kopfes.



**Original** w=1 (krümmungs**un**abhängig) w=2.5 w=5 w=10 (stark krümmungsabhängig)

**Abbildung 2-4: Einfluss der Krümmungs-Gewichtung auf die Triangulation**

Es ist deutlich ersichtlich, dass das ganze 'Nesting Birds'-Verfahren durch die Datenreduktion zwangsläufig unvorhersehbare, mehr oder minder entstellende Vereinfachungen am Objekt vornimmt. Durch die krümmungsabhängige Verteilung bleibt der Detailgehalt jedoch in den Bereichen hoch, die gemeinhin als interessant gelten. Das ganze Verfahren ist zwar deterministisch (reproduzierbare Resultate bei Wiederholung mit gleichen Parametern) aber kleine Parameteränderungen können das Triangulationsresultat für den Betrachter wesentlich verändern.

## 2.4. Probleme

Die krümmungsabhängige Verteilung führt in den untersuchten Fällen zu keinen sichtbaren Artefakten. Es ist jedoch denkbar, dass unter bestimmten Bedingungen solche auftreten können.

Durch die künstliche Erhöhung der Punktdistanzen bei hoher Krümmung können nahe Punkte plötzlich weiter weg erscheinen als entferntere. Das könnte dazu führen, dass die Topologieinformation, die implizit aus den Distanzen gewonnen wird, nicht mehr mit der Topologie übereinstimmt, die durch die Punktkonnektivitäten explizit vorgegeben ist. Das Problem wird möglicherweise etwas abgeschwächt durch das Vorgehen bei der Distanzberechnung. Von jedem Punkt aus werden der Topologie folgend nur solange Distanzen zu den Nachbarpunkten berechnet, bis eine vorgegebene Maximaldistanz erreicht ist. Bei grossen Krümmungen wird darum auch die Distanzberechnung vorzeitig abgebrochen.

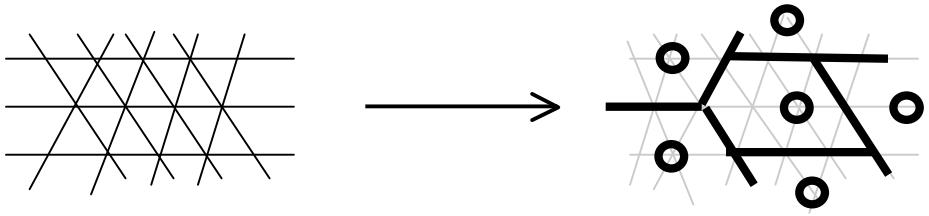
Das zweite Problem ist, dass die krümmungsabhängige Verteilung unvermeidlich auch uninteressanten Artefakten in den Ausgangsdaten ein grösseres Gewicht verleiht. In der Praxis wird aber auch dieses Problem abgeschwächt, weil sich offensichtlich kleine Artefakte, die sich auf einzelne Messpunkte beziehen (z.B. Mess-Rauschen) nicht durchsetzen können. Zu untersuchen wäre, ob einzelne Messpunkte, die sehr weit von ihrem effektiven Wert abweichen zu Artefakten führen können.

Weiter wäre es interessant, das Verhalten von 'rauhen' Oberflächen zu untersuchen, bei denen sehr kleinräumig hohe Krümmungen auftreten. Dort müsste vielleicht nicht die lokale Krümmung zwischen den zwei betrachteten Punkten sondern eine über die Umgebung gemittelte Krümmung verwendet werden. Der Rechenaufwand für diese Variante dürfte aber beträchtlich grösser sein.

# 3. Triangulation

## 3.1 Aufgabenstellung

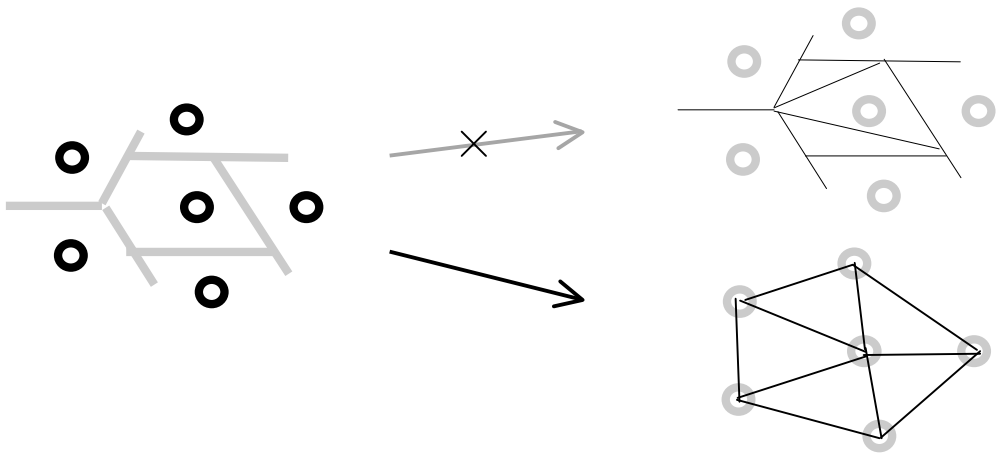
Das von Cyrille Albrecht und Christoph Zollikofer entwickelte 'Nesting Birds'-Verfahren zur Polygonreduktion von Freiformoberflächen wurde in der Diplomarbeit von C. Albrecht so weit implementiert, dass das Triangulationsnetz der Originaloberfläche in ein gröberes (reduziertes) Polygonnetz (sog. Patches) umgewandelt wurde. Dabei stammen alle Patchmittelpunkte im reduzierten Modell aus dem Original. Das Verfahren generiert also keine neuen Stützpunkte, sondern benutz nur Punkte aus dem Originaldatensatz.



Original-Triangulation Reduzierung auf Patchmittelpunkte und Patches

**Abbildung 3-1: Schematische Darstellung der Vorarbeit von C. Albrecht**

Als nächster Schritt soll daraus wieder eine triangulierte Oberfläche abgeleitet werden, da Dreiecke einfach visualisierbar und behandelbar sind. Dreiecke haben gegenüber Polygonen den Vorteil, dass ihre Flächennormale und ihr Flächeninhalt einfach bestimmbar sind. Da im 'Nesting Birds'-Verfahren mit viel Aufwand die Patchmittelpunkte bestimmt werden, die Patches darum herum aber nur untergeordnete Hilfsmittel zur Visualisierung sind, sollen auch für die abschliessende Triangulation die Mittelpunkte als Stützpunkte verwendet werden und nicht (was einfacher wäre) die Patches in Dreiecke zerlegt werden (vgl. Abbildung 3-2).



Reduzierung auf Patchmittelpunkte Angestrebte Re-Triangulation

**Abbildung 3-2: Schematische Darstellung des Ziels dieser Arbeit**

## 3.2. Lösungsansatz

Es erscheint wenig ratsam, die Patchmittelpunkte, welche die Stützpunkte unserer Triangulation bilden sollen, aus ihrem bisherigen Kontext herauszulösen und von Grund auf eine neue Triangulation zu machen. Die enstehende Triangulation würde der konvexen Hülle der Patchmittelpunkte entsprechen, während die Ausgangsoberfläche weder geschlossen noch zuammenhängend noch konvex gewesen sein muss.
Vielmehr wollen wir versuchen, die vorhandene topologische Information zu nutzen und benutzen dazu eine Untermenge der Originalpunkte, nämlich die bereits ermittelten Ränder der Patches (vgl. Abb. 3-1). Indem wir die Patchrandpunkte betrachten, die zu mehr als 2 Patches gehören (sog. Kreuzungspunkte), können wir die relative Lage benachbarter Patchmittelpunkte ermitteln und jeweils drei davon als Ecken für die neuen Dreiecke auswählen.

## 3.3. Algorithmus

### 3.3.1. Voraussetzung

Gegeben sind a) eine zusammenhängende triangulierte Struktur, b) eine Menge ausgewählter Punkte (“Patchmittelpunkte”) derselben, die zusammen die neue reduzierte Struktur bilden und über die eine neue Triangulation gemacht werden soll und c) eine Zuordnung aller Punkte zu einem oder mehreren Patchmittelpunkten.

### 3.3.2. Importieren der Daten

Die Ausgangsdaten werden in zwei simplen Liste gesammelt, die die gesamte Topologie widerspiegeln. Die erste (Abbildung 3-1) enthält zu jedem Patchmittelpunkt die zugehörigen Patchrandpunkte nach Umlaufsinn geordnet. Dabei werden einfache Plausibilitätsprüfungen gemacht. So werden aufeinanderfolgende identische Randpunkte eliminiert und ebenso ganze Patches, die weniger als drei Randpunkt haben.

```
M1:     R1, R2, R3, ..
M2:     R2, R4, R5, ..
…                                   (M=Mittelpunkt, R=Randpunkt)
```

**Abbildung 3-1: Datenstruktur 1 (Mittelpunkte->Randpunkte)**

Das Gegenstück ist eine Liste (Abbildung 3-2), die die Zugehörigkeit von jedem Randpunkt zu einem oder mehreren Mittelpunkten festhält und somit die relative Lage der Patches zueinander beschreibt.

```
R1:     M1, ..
R2:     M1, M2, ..
…                                   (M=Mittelpunkt, R=Randpunkt)
```

**Abbildung 3-2: Datenstruktur 2 (Randpunkte->Mittelpunkte)**

Dabei können die gleichen Patchrandpunkte zu verschiedenen Patchmittelpunkten gehören (z.B. R2 ist Randpunkt zu M1 und M2) und es kann auch ein Patchmittelpunkt gleichzeitig ein Patchrandpunkte sein.

### 3.3.3. Topologie der Randpunkte ermitteln

Für die Triangulation gehen wir von denjenigen Patchrandpunkten aus, die zu mehr als 2 Patchmittelpunkten gehören und somit Knotenpunkte K zwischen mindestens drei Patches sind.
Von diesen Knotenpunkten ausgehend wird die Triangulation zwischen den Mittelpunkten der angrenzenden Patches ausgeführt. Dazu wird zuerst die Topologie um den Knotenpunkt herum untersucht.
Für jeden angrenzenden Patch $M_i$ wird passend zu seiner Orientierung (Patch-Drehsinn) die Abfolge der Randpunkte festgestellt, bzw. die Orientierung der Patchränder relativ zum Knotenpunkt notiert.

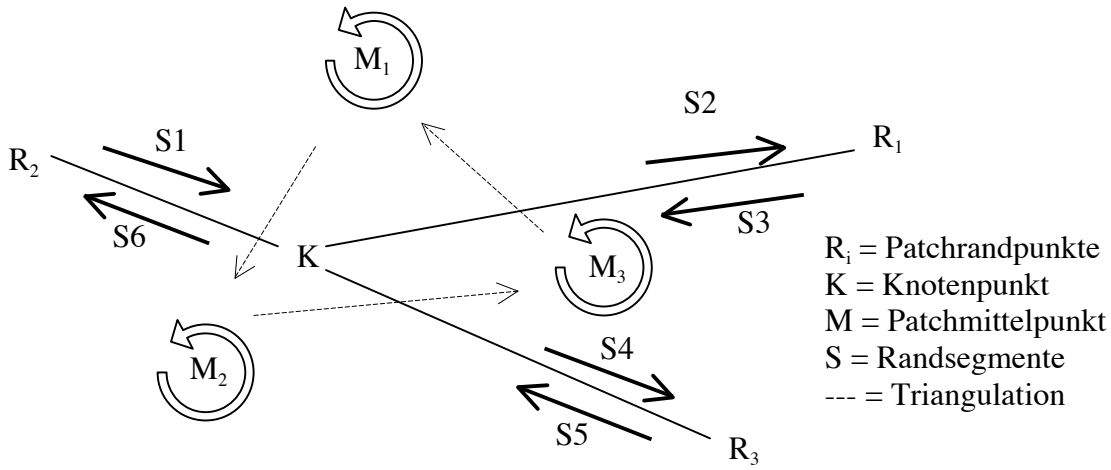



**Abbildung 3-5: Topologie der Randpunkte**

In der Abbildung 3-6 betrachtet man beispielsweise die Umgebung von Knotenpunkt K, der ein Randpunkt zu den Patchmittelpunkten M1, M2 und M3 ist. Dem Patch-Drehsinn folgend lassen sich die Randpunkte (R) und Randsegmente (S) der zu K benachbarten Patches ordnen:

| | | |
|---|---|---|
| M1: | R2 K R1 | bzw. S1 S2 |
| M2: | R3 K R2 | bzw. S5 S6 |
| M3: | R1 K R3 | bzw. S3 S4 |

(M=Mittelpunkt, R=Randpunkt, K=betrachteter Knotenpunkt, S=Segment)

**Abbildung 3-6: Umgebung um Punkt K: Topologie der Randsegmente**

Aus dieser Information kann zu jedem Patch die relative Lage der angrenzenden Patches ermittelt werden, indem zusammenpassende Randsegmente gesucht werden. In unserem Beispiel unterscheiden sich S1 und S6, S2 und S3 sowie S4 und S5 nur durch ihre Richtung und bilden so Paare, mit deren Hilfe sich nun für jeden Patch die relative Lage der Nachbarspatches angeben lässt:

| | | |
|---|---|---|
| M1: | M3 M1 M2 | d.h. M1 liegt zwischen M3 und M2 |
| M2: | M1 M2 M3 | d.h. M2 liegt zwischen M1 und M3 |
| M3: | M2 M3 M1 | d.h. M3 liegt zwischen M2 und M1 |

(M=Mittelpunkt)

**Abbildung 3-7: Umgebung um Punkt K: Topologie der Mittelpunkte**

Daraus lässt sich ein Graph der Topologie ableiten, der in unserem einfachen Beispiel bereits der Triangulation entspricht.


**Abbildung 3-8: Umgebung um Punkt K: Graph der Gesamttopologie**

Der ganze Aufwand muss deshalb getrieben werden, weil die Topologie um einen Knotenpunkt sehr komplex sein kann. Es muss damit gerechnet werden, dass eine Kante zu mehr als 2 Patches gehört oder auch nur zu einem (Knoten liegt auf dem Rand). Das folgende kompliziertere Beipiel zeigt, wie das vorgestellte Vorgehen auch solche topologische Spezialfälle bewältigt.

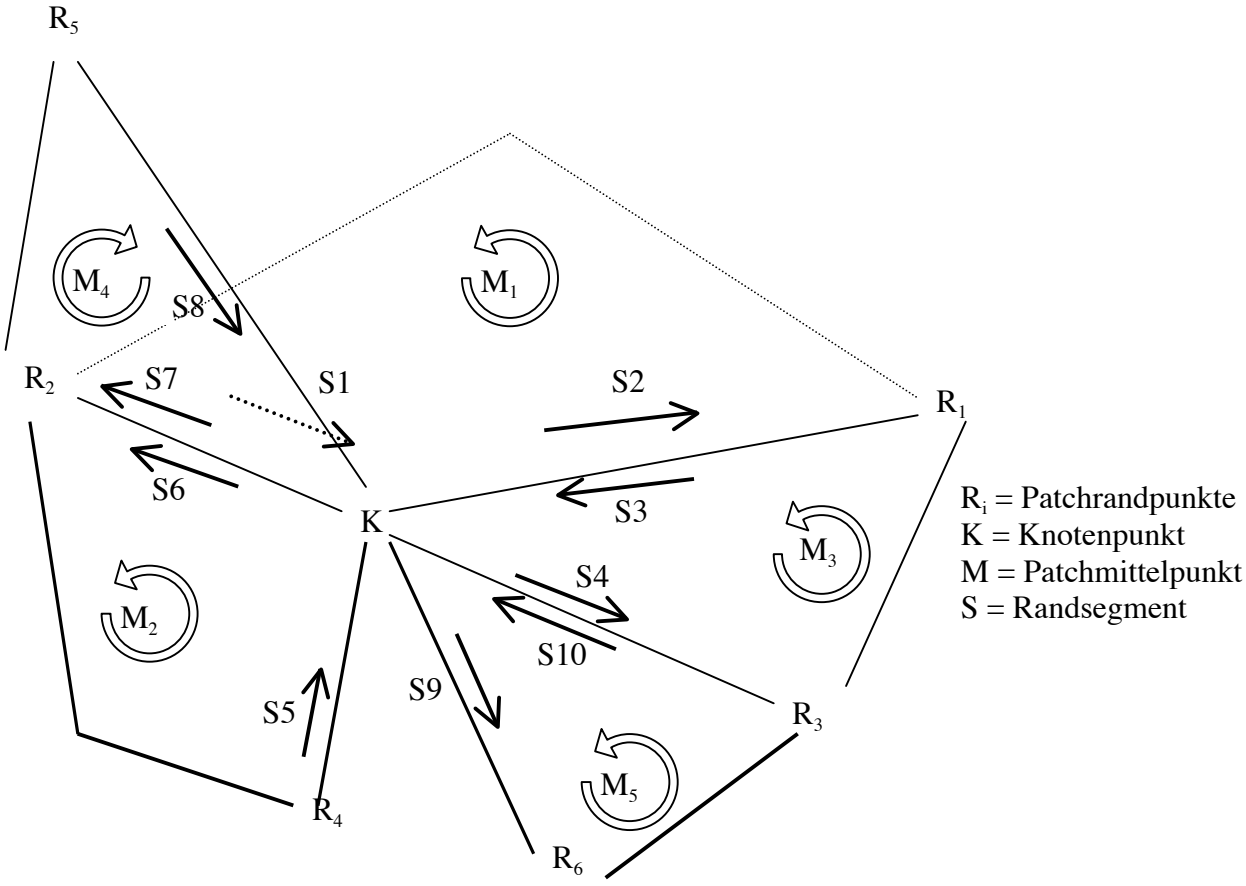



**Abbildung 3-9: Möglicher degenerierter Fall**

Zwichen den Randsegmenten S5 und S9 haben wir hier ein Loch in der Struktur (oder eine Aussenkante) und zwischen K und R2 stossen drei Patches aufeinander. Die Kantenorientierung um K herum ist in Abbildung 3-10 aufgeführt. Das ergibt die in Abbildung 3-11 angegebene Lage der Nachbarpatches zueinander und den Topologiegraphen in Abbildung 3-12. Hier ist die Ableitung der Triangulation aus dem Topologiegraphen nicht mehr so einfach wie in Abbildung 3-8.

| | | |
|---|---|---|
| M1: | R2 K R1 | bzw. S1 S2 |
| M2: | R4 K R2 | bzw. S5 S6 |
| M3: | R1 K R3 | bzw. S3 S4 |
| M4: | R5 K R2 | bzw. S8 S7 |
| M5: | R3 K R6 | bzw. S10 S9 |

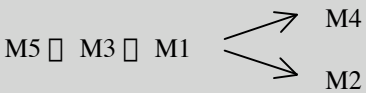
(M=Mittelpunkt, R=Randpunkt, K=betrachteter Knotenpunkt, S=Segment)

**Abbildung 3-10: Degenerierter Fall: Topologie der Randsegmente**

| | | |
|---|---|---|
| M1: | M3 M1 M4, M2 | d.h. M1 liegt zwischen M3 und M2/M4 |
| M2: | M1 M2 | d.h. M2 liegt 'hinter' M1 |
| M3: | M5 M3 M1 | d.h. M3 liegt zwischen M5 und M1 |
| M4: | M1 M4 | d.h. M4 liegt 'hinter' M1 |
| M5: | M5 M3 | d.h. M3 liegt 'hinter' M5 |

(M=Mittelpunkt)

**Abbildung 3-11: Degenerierter Fall: Topologie der Mittelpunkte**




(M=Mittelpunkt)

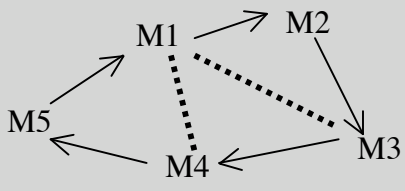
**Abbildung 3-12: Degenerierter Fall: Graph der Gesamttopologie**

## 3.3.4. Triangulation

Im letzte Schritt muss aus dem Topologiegraphen eine Triangulation abgeleitet werden. Dabei wird so vorgegangen, dass häufig auftauchende und einfach zu behandelnde Fälle identifiziert werden. Es zeigt sich nämlich, dass wir es in unserem Anwendungsgebiet (Computertomogramme von fossilen Knochen) meist mit geschlossenen Oberflächen oder zumindest mit solchen ohne 'Verzweigungen' im Topologiegraphen zutun haben. Wir behandeln darum nur die Fälle der geschlossenen und offenen unverzweigten Graphen.

### 3.3.4.1. Geschlossene unverzweigte Graphen

Dieser Fall tritt sehr häufig auf (zB in Abbildung 3-5) und lässt sich einfach behandeln. Falls der Graph nur 3 Knoten hat (Abbildung 3-8) ist die Ableitung trivial: es resultiert ein Dreieck mit den Knoten des Graphen als Ecken. Bei mehr als 3 Knoten wird ausgehend von einem beliebigen Knoten in Dreiecke aufgeteilt (Abbildung 3-13). Für n>2 Knoten ergeben sich somit n-2 Dreiecke mit den Ecken ($M_1$, $M_{i-1}$, $M_i$ ; i=2..n).




(M=Mittelpunkt)

**Abbildung 3-13: Triangulation von geschlossenen unverzweigten Graphen**

### 3.3.4.2. Offene unverzweigte Graphen

Patches, die am Rand einer Struktur liegen oder die an ein Loch in der Struktur grenzen, haben nicht auf allen Seiten Nachbarpatches. Dadurch entstehen nicht-geschlossene Topologiegraphen. Sie lassen sich ebenfalls einfach behandeln, indem die zusammenhängenden Segmente jeweils nach dem selben Verfahren wie bei geschlossenen Graphen trianguliert werden (Abbildung 3-14). Die Einteilung in Dreiecke muss hier jedoch jeweils von den Anfangspunkten der Segmente aus erfolgen (M1 und M4).

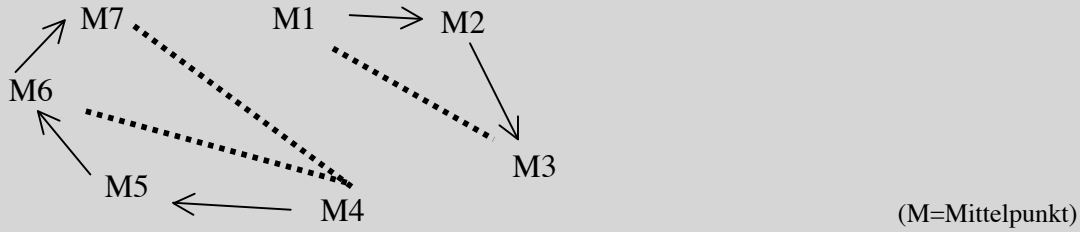



**Abbildung 3-14: Triangulation von offenen unverzweigten Graphen**

### 3.3.4.3. Verzweigte Graphen

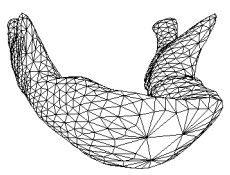
Bei den untersuchten Beispielen traten verzweigte Graphen praktische nie auf und wurden darum ignoriert. Theoretisch könnte man innerhalb dieser Graphen nach Zyklen oder nach den längsten zusammenhängenden Stücken suchen und diese Segmente wie unverzweigte Graphen behandeln.

## 3.4. Resultate

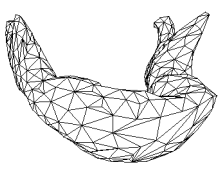
Der Anspruch des 'Nesting Birds'-Algorithmus, bei möglichst guter Beibehaltung der Originalform und -geometrie die Anzahl der Polygone zu reduzieren, und die Qualität der Triangulation müssen sich letztlich am Auge des Betrachters messen. Darum soll an einigen Beispielen untersucht werden, welche Resultate die Triangulation erzielt. (Für ein weiteres Beispiel vgl. auch Abb. 2-4!)

### 3.4.1. Datensatz 'Kieferknochen'

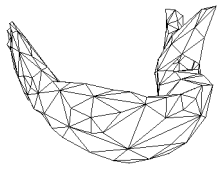
Das erste Beispiel zeigt eine geschlossene Oberfläche (ein Kieferknochen) bei verschiedenen Kompressionsstufen. Kompressionsstufen "5" bedeutet hierbei, dass das Objekt in seiner längsten Achse in etwa 5 Dreiecke unterteilt wird. Bis zur Stufe "10" wird das Original sehr gut reproduziert. Es sind keine Artefakte sichtbar.



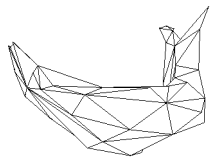
**Original**
**Punkte:** 1048
**Dreiecke:** 2092



**Reduktion "20"**
**Patches:** 383
**Dreiecke:** 737



**Reduktion "10"**
**Patches:** 171
**Dreiecke:** 328



**Reduktion "5"**
**Patches:** 62
**Dreiecke:** 99

**Abbildung 3-15: Triangulationsbeispiel: 'Kieferknochen'**

### 3.4.2. Datensatz ‘Fläche’

Das zweite Beispiel zeigt einen Ausschnitt aus einer Knochenoberfläche (nicht-geschlossenes Objekt). Wiederum werden bis Stufe “10” akzeptable Resultate erzielt. Für die Reduktionsstufen “5” und “10” ist auch ersichtlich, wie die Dreiecke aus der Patch-Einteilung abgeleitet worden sind. Das Loch in der Triangulation bei der Reduktionsstufe “10” lässt sich auf einen Fehler in der Patchgenerierung zurückführen. Die Patchstruktur hat dort ebenfalls ein Loch (grau in Abb. 3-16).

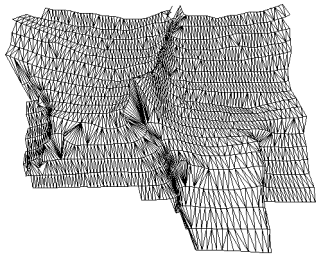
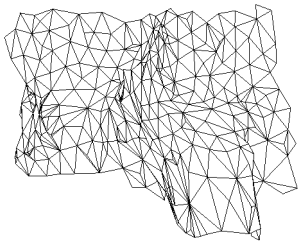
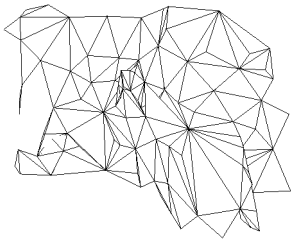
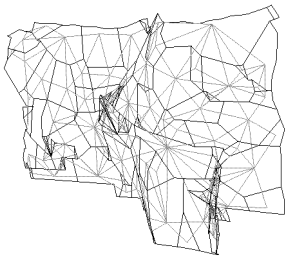
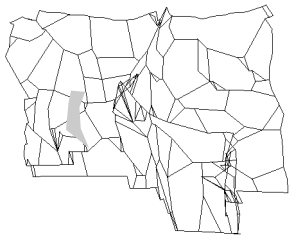
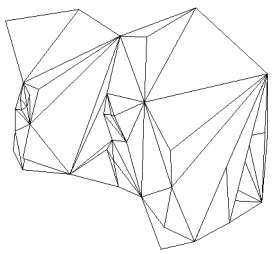
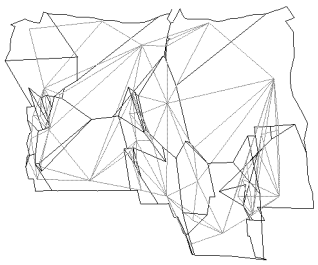
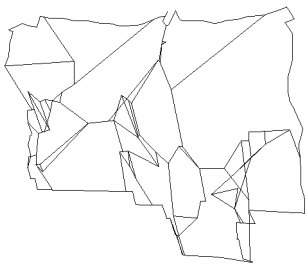
| | | | |
|---|---|---|---|
| **Original** |  **Punkte:** 1802 **Dreiecke:** 3403 | | |
| **Reduktion “20”** |  **Dreiecke:** 583 (**Patches:** 343) | | |
| **Reduktion “10”** |  **Dreiecke:** 189 |  **Dreiecke:** 189 Patches: **144** |  **Patches:** 144 |
| **Reduktion “5”** |  **Dreiecke:** 70 |  **Dreiecke:** 70 Patches: **67** |  **Patches:** 67 |
| | **Triangulation** | **Triangulation & Patches** | **Reduzierte Patches** |

**Abbildung 3-16: Triangulationsbeispiel: ‘Fläche’**

## 3.5. Diskussion

### 3.5.1. Bewertung

Die vorgestellte Triangulationsmethode liefert für die untersuchten Fälle gute Resultate. Nur bei sehr starker Reduktion sind Artefakte sichtbar, die aber nicht direkt auf die Triangulation zurückzuführen sind. Die Methode ist effizient genug, um auch auf wesentlich grössere Datenmengen angewandt werden zu können. Durch die sequentieller Abarbeitung der Kreuzungspunkte wäre sogar eine Parallelisierung des Algorithmus einfach realisierbar.
Es zeichnen sich aber verschiedene Punkte ab, die noch verbessert werden müssen, um zuverlässige Resultate zu erhalten.

### 3.5.2. Degenerierter Input

Der Schritt von den Patchmittelpunkten zu den Patches (Abb. 3-1, rechts) ist zuwenig zuverlässig und liefert teilweise degenerierte Daten, die bei Triangulation wieder herausgefiltert werden müssen. Insbesonderen zeigen sich folgende Artfakte:

- Patches mit einem oder zwei Ecken treten auf.
- Patchmittelpunkte kommen oft nicht in der Mitte eines Patches zu liegen, sondern treten gehäuft in einer bestimmten Ecke des Patches auf (graue Kreise in Abb. 3-17).
- Es treten Löcher auf, weil zwar alle Originalpunkte zu einem Patchmittelpunkt zugeordnet werden, aber nicht alle Originaldreiecke Teil einer Patchfläche geworden sind (Abb. 3-17). Dies rührt daher, dass bei der Patchgenerierung mit Punkten statt Dreiecken gearbeitet wird.

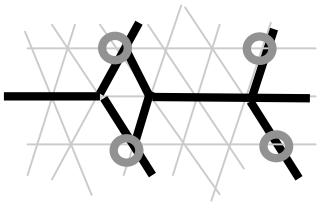


**Abbildung 3-17: Degenerierter Input: Löcher und verschobene Mittelpunkte**

Die Generierung der Patches aus den Patchmittelpunkten müsste neu programmiert werden. Folgende Strategie könnte dabei angewandt werden:

- Von jedem Mittelpunkt aus werden alle Nachbardreiecke angefärbt.
- Falls eine Dreieck dabei an mehrere Mittelpunkte grenzt, wird es in zwei oder drei Dreiecke unterteilt.
- Alle übrigen Dreiecke werden möglichst gleichmässig auf die Patches verteilt, indem iterativ jedem Patch diejenigen Dreiecke provisorisch zugeteilt werden, die mit mindestens einer Seite an den Patch angrenzen. Wenn bei einer Iteration ein Dreieck zu mehreren Patches zugeteilt wird, muss es wie im vorherigen Schritt unterteilt werden

Damit würde erreicht, dass Mittelpunkte nie auf den Patchrand zu liegen kommen und keine Löcher entstehen.

Bei der jetzigen Implementation wird nachträglich versucht, bei der Triangulation diese Probleme zumindest teilweise zu korrigieren. In allen gezeigten Abbildungen wurde die Lage der Mittelpunkte nachträglich als geometrisches Mittel der Patchrandpunkte interpoliert, was zwar an der Topologie nichts ändert aber den optischen Eindruck stark verbessert (Abb. 3-18). Dadurch wird allerdings eine der Triangulationsvorgaben verletzt, die fordert, dass wir nur Punkte benutzen, die im Originaldatensatz vorhanden sind. Bei geschlossenen, konvexen Objekten vermindert sich durch die Interpolation auch das Volumen des Objektes beträchtlich (vgl. auch Abb. 3-18). Die

Interpolation liesse sich verbessern, indem nicht das geometrische Mittel als Mittelpunkt genommen wird sondern der Punkt innerhalb des Patches, der den kleinsten Abstand zum geometrischen Mittelpunkt hat. Es ist aber wohl sinnvoller, das Problem bei der vorgelagerten Patchgenerierung zu lösen statt im Nachhinein zu korrigieren.

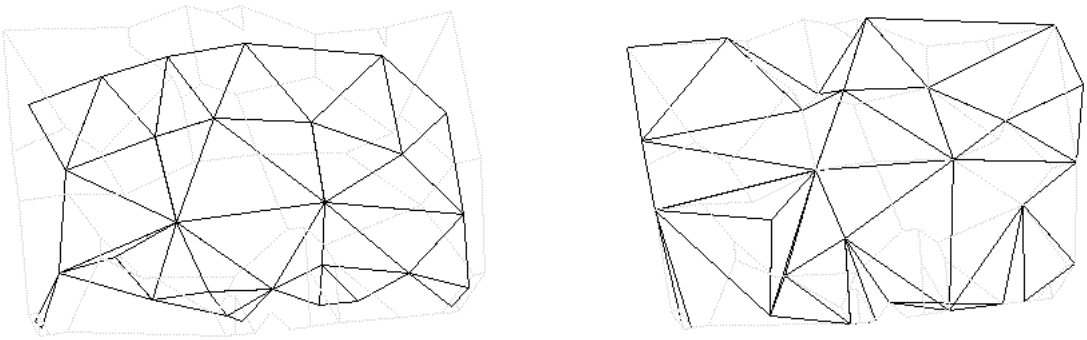


**Abbildung 3-18: Interpolierte (links) und nicht-interpolierte Mittelpunkte**

### 3.5.3. Konkave Patches

Ein ganz anderes Problem ergibt sich bei der Umwandlung des Topologiegraphen in Dreiecke (Kap. 3.3.4.). Insbesondere wenn die Patches konkav sind, müsste für die Umwandlung auch die Position der Mittelpunkte und nicht nur ihre relative Lage zueinander betrachtet werden. Schon für die einfache Topologie in Abbildung 3-8 kann es Sonderfälle geben, die zu unschönen Artefakten führen. Wenn wie in Abbildung 3-19 der Mittelpunkt von Patch 1 statt bei M1 bei M1' liegt, ändert sich nichts an der Topologie um den Kreuzungspunkt K, weil bei der Topologiebestimmung die Lage der Mittelpunkte innerhalb der Patches keine Rolle spielt. Das resultierende Dreieck (Pfeile in Abb. 3-19) bekommt aber dabei eine andere Orientierung. Die entstehenden Artefakte sehen aus wie Überwerfungen, sind aber meist nur lokaler Natur.

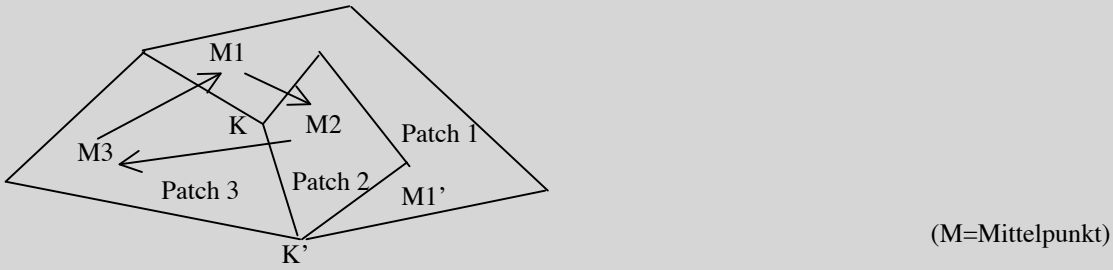



**Abbildung 3-19: Topologie und problematische Geometrie bei konkaven Patches**

Ein zusätzliches mögliches Problem ist ebenfalls in Abbildung 3-19 sichtbar. Es könnte vorkommen, dass zwei Kreuzungspunkte K und K' zwischen den selben Dreiecken liegen oder es könnte sogar ein konkaver Patch mehrfach an den selben Kreuzungspunkt angrenzen. Wiederum ist dies eine Fehlerquelle, die weitgehend durch eine gute Patchgenerierung eliminiert werden kann.

### 3.5.4. TDF-Ausgabe

Verbesserungswürdig ist schliesslich die Ausgabe der Triangulationsdaten im TDF-Format. TDF würde es erlauben, Punktkoordinaten, die zu mehreren Dreiecken gehören, nur einmal auszugeben und von mehreren Dreiecken aus zu referenzieren. Momentan werden Punktkoordinaten redundant für jedes Dreieck separat geschreiben. Dadurch werden die Ausgabedateien grösser als nötig.

# 4. Zusammenfassung

In dieser Semesterarbeit wurde die Diplomarbeit von C. Albrecht [Albrecht2000] um zwei wesentliche Teile erweitert.
Die Polygonreduktion kann nun krümmungsabhängig erfolgen (Kap. 2.), was für den Betrachter die Ähnlichkeit des datenreduzierten Objekts zum Original erhöht (Abbildung 2-4).
Zur Rückführung des datenreduzierten Objekts in eine triangulierte Form wurde eine einfache aber effektive Methode entworfen (Kap. 3.) und implementiert (Anhang A), die für die betrachteten Daten sehr gute Resultate liefert aber einige Grenzfälle noch nicht abdeckt. Die Mehrzahl der auftretenden Triangulations-Artefakte (Kap. 3.5.) lässt sich vermutlich durch eine verbesserte Patchgenerierung vermeiden.

Sehr lehrreich war es, sich mit relativ komplexem fremdem Sourcecode auseinandersetzen zu müssen, der zwar auf Prozedur-Ebene recht gut dokumentiert war, der es einem aber nicht leicht machte, das Zusammenspiel dieser Einzelteile zu begreifen. Es war sehr nützlich, ein Gefühl für den Aufwand für solche Maintenance-Aufgaben zu bekommen und am eigenen Leib zu erfahren, was eine Dokumentation beinhalten muss, damit eine Drittperson die Arbeit weiterführen kann. Mit Anhang A hoffe ich, einem etwaigen Nachfolger das Weiterführen dieser Arbeit erleichtert zu haben.

# 5. Dank

Diese Arbeit entstand am Institut für Informatik der Universität Zürich im Multimedialabor (MML). Mein Dank gilt dessen Leiter, Prof. Dr. Peter Stucki, für die interessante Aufgabenstellung und meinem Betreuer PD Dr. Christoph Zollikofer, der mich mit viel Geduld in die Materie eingeführt hat und mir mit guten Ideen unermüdlich zur Seite gestanden ist.

# 6. Literaturverzeichnis

[Albrecht2000] Albrecht, C.
Nesting Birds - Eine neue Methode zur Polygonreduktion von Freiformoberflächen,
Diplomarbeit am Institut für Informatik der Universität Zürich, 2000

# 7. Abbildungsverzeichnis

# Anhang A – Implementation

Dieser Anhang versucht, eine Brücke zwischen Kapitel 3.3. (Triangulationsalgorithmus) und dem Programmcode zu schlagen. Dadurch soll eine Weiterführung dieser Arbeit erleichtert werden.
Als Sprache wurde C++ verwendet. Zur Bezeichnung von Funktionen und Klassen wird hier die Notation **KlassenName::funktionsName()** verwendet. Die Klasse 'KlassenName' findet sich jeweils in den Dateien 'KlassenName.h' und 'KlassenName.cpp'. (Zur Vereinfachung wird in der Beschreibung bewusst auf eine Differenzierung zwischen Klassen und Objekten verzichtet.)

## 1. Programmablauf

Die Funktion **Schnittstelle**::**run()** steuert den Programmablauf, indem sie folgende Aktionen auslöst:

- **Schnittstelle::ladeParameterVonDatei()** liest den Namens der zu reduzierenden TDF-Datei und die Reduktionsparameter aus der Datei 'Parameter.txt'.
- **Daten::ladeDatenAusTDF()** liest die zu reduzierenden TDF-Datei in die Klasse **Daten** ein.
- **Reduktor::erstelleVerteilung()** nimmt für die übergebenen **Daten** die Verteilung der Patchmittelpunkte nach dem 'Nesting Birds'-Algorithmus vor.
- **Reduktor::erstelleReduktion()** ermittelt die Patches um die Patchmittelpunkte herum und speichert sie in der Klasse **Patchverwaltung** ab.
- **Triangulator::erstelleTriangulation()** führt aufgrund der Information in den Klassen **Daten** (Koordinaten der Punkte) und **Patchverwaltung** (Topologie der Patchmittelpunkte und -randpunkte) die Triangulation durch und gibt das Resultat im TDF-Format aus.
- Schematische Darstellung:

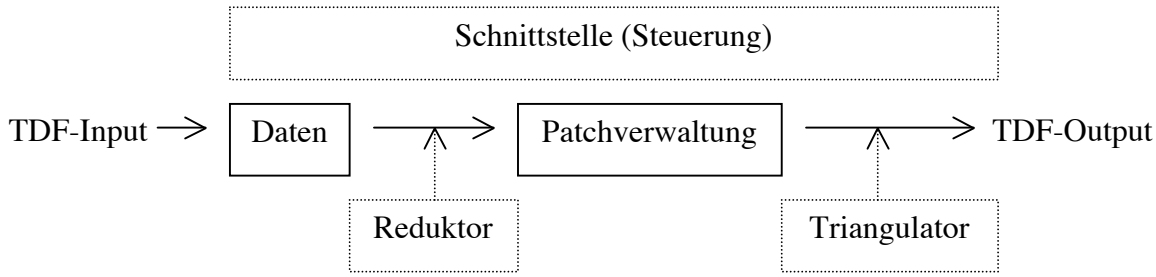



**Abbildung A-1: Schematische Programmstruktur**

## 2. Hilfsklassen

Zusätzlich werden folgende Hilfsklassen verwendet:

- **Item**: von Klasse **Reduktor** intern benutzt.
- **Punkte**: von den Klassen **Reduktor** und **Patchverwaltung** benutzt.
- **array_list**, **fi3_ff** und **triangle_list**: benutz innerhalb von der Klasse **Daten** und zum TDF-Import/-Export.

## 3. Implementation der Krümmungsabhängigen Verteilung

- Die krümmungsgewichtet Distanz wird innerhalb von **Reduktor::erstelleVerteilung()** beim Schritt **Reduktor::FindeNachbarn()** benötigt. Die Distanzberechnung wird in

objektorientierter Manier dorthin delegiert, wo die Daten liegen, nämlich ins **fi3_ff** –Objekt, das von **Daten** verwaltet wird.
- Dadurch wird **fi3_ff:find_weighted_neighbour_vertices()** aufgerufen, um alle Punkte zu finden, die innerhalb eines vorgegebenen Abstands um einen gewählten Punkt liegen.
- Die eigentliche Distanzberechnung und -gewichtung findet in **fi3_ff:weighted_dist_squared()** statt.

# 4. Implementation der Triangulation

Die gesamte Triangulation findet innerhalb der Klasse **Triangulator** statt. Diese Klasse verwaltet eine dynamische Listen ***patchMittelPunkt,*** welche für jeden Mittelpunkt eine dynamischen Liste der Randpunkte enthält und eine Liste ***patchRandPunkt***, die umgekehrt zu den Randpunkten die Mittelpunkte enthält.

Gegen aussen ist nur die Funktion **erstelleTriangulation** sichtbar, welche die Triangulation steuert. Sie nimmt als Parameter die Koordinaten der Punkte (**Daten**) und die Topologie der Patchmittelpunkte und –randpunkte (**Patchverwaltung**).
Danach führt sie folgende Schritte durch:

- **cleanPatchverwaltung()** eliminiert direkt in der Patchverwaltung offensichtlich fehlerhafte Daten (Patches mit mehreren gleichen oder mit weniger als drei Randpunkten).
- **allozierePunktZuPatchListe()** stellt Speicher bereit für den Import der Topologiedaten in die Listen ***patchMittelPunkt*** und ***patchRandPunkt***.
- **erstellePunktZuPatchListe()** importiert Patches aus der **Patchverwaltung** und sortiert sie mittels **fuegePunktZuPatch()** in die beiden Listen ***patchMittelPunkt*** und ***patchRandPunkt*** ein. Jeder Patchmittelpunkt und Patchrandpunkt wird nur als Verweis auf eine globale Punkteliste gespeichert.
- **generiereMarkierpunktVerbindungen()** ruft für jeden Randpunkt in ***patchRandPunkt***, der zu mehr als 2 Mittelpunkten gehört, die Prozedur **untersucheMehrfachEcke()** auf.
- **untersucheMehrfachEcke()** sammelt mit Hilfe von **getNextEdgeOfPatch()** und **getPreviousEdgeOfPatch()** Informationen über die Umgebung der Mehrfach-Ecke und klassifiziert die dabei gewonnenen Topologie-Graphen. Für die einfach behandelbaren Fälle wird dann **triangulate()** aufgerufen, das Dreiecke bildet und in der ***edgeList*** zwischenspeichert.
- **writeTrianglesToTDF()** schreibt die in ***edgeList*** gepeicherten Dreiecke im TDF-Format in eine Datei.
- **loeschePunktZuPatchListe()** löscht die nicht mehr benötigten Listen ***patchMittelPunkt*** und ***patchRandPunkt***.

# Anhang B – Was sonst noch geschah…

Im Hauptteil dieser Semesterarbeit wurde derjenige Teil der Arbeit beschreiben, der Anspruch auf einen gewissen 'wissenschaftlichen' Wert erhebt. Um diese Arbeit zu ermöglichen, mussten aber auch umfangreiche Vorarbeiten geleistet werden, die es verdienen, zumindest im Sinne eines kurzen Rechenschaftsberichts erwähnt zu werden.

### Portierung

In seiner Diplomarbeit hat C. Albrecht den 'Nesting Bird'-Algorithmus mit Visual C++ 6.0 auf Windows NT 4.0 implementiert. Eine Visualisierung wurde durch VTK und eine Ausgabe für das 3D-Programm Maya ermöglicht. Das Programm wurde auf Mac OS X 10.1 portiert (Programmierumgebung ProjectBuilder mit GCC Compiler 2.95 und später 3.1).

### Visualisierung

Graphische Algorithmen lassen sich zwar 'mit Papier und Bleistift' konzipieren. Um die Qualität einer Triangulation zu überprüfen oder Fehler in der Topologie zu entdecken kommt man um eine Visualisierung jedoch nicht herum. Für das 3D-Dateiformat TDF von Ch. Zollikofer wurde auf der Basis von GLUT (einer Bibliothek die auf OpenGL aufsetzt) ein Viewer geschrieben, der folgende Eigenschaften unterstütz:

- Darstellbare Objekte: Punkte, Linien und Dreiecke
- Rotation/Grössenänderung von Objekten
- Einmittung von Objekten durch Bounding-Box-Berechnung
- Darstellung von Flächen als Wireframe, Hidden-Line, Flat-shaded und Smooth-shaded (Interpolation zwischen Vertex-Normalen)
- Darstellung von Vertex- und Flächennormalen
- Übereinanderlegen mehrerer Objekte und farbliche Darstellung von Objektattributen
- Optionales Antialiasing von Linien

Alle Abbildungen von dreidimensionalen Objekten in dieser Arbeit sind Screenshots von diesem Programm. Die Visualisierung hat sich als unentberliches Hilfsmittel erwiesen, um Fehler in der Triangulation zu finden.

### Erweiterung am Programm

Das Programm von C. Albrecht wurde in der Grobstruktur unverändert belassen. Einige wenige Fehler im Code sind korrigiert worden. Darüber hinaus wurden folgende Erweiterungen vorgenommen:

- Die Abhängigkeiten zwischen den verschiedenen Programmteilen wurde reduziert (vgl. Abbildung A-1). Die C++ - Klassen wurden so rigoros wie möglich mit Zugriffsrechten (public/private) versehen, um eine sauberere Modularisierung zu fördern.
- Es wurde ein abgestuftes Logging-System eingeführt, um selektiv nur Fehlermeldungen oder auch Diagnose-Output anzeigen zu können. Der Log-Level ist als Programmparameter ohne Neukompilierung einstellbar.

- Ein einfaches Timing der einzelnen Schritte (Verteilung/Reduktion/Triangulation/ Speichern) wurde eingeführt. Die Rechenzeit der Triangulations-Phase erweist sich dabei gegenüber den Schritten ‘Verteilung’ und ‘Reduktion’ als vernachlässigbar.
- Der Kompressionsgrad lässt sich beim ‘Nesting Birds’-Algorithmus dadurch steuern, dass die Distanz festgelegt wird, bis zu welcher Nachbarpunkte gesucht werden sollen. Dieser Parameter ist unglücklicherweise von der Skalierung des Objekts abhängig und erfordert, dass man die absolute Grösse des Objektes kennt. Es wurde darum eine relative Distanzangabe eingeführt: für negativen Distanzangaben (-D) berechnet sich die Distanz, indem die Raumdiagonale der Bounding-Box des Objekts durch D geteilt wird.