Media Informatics /
Graphics & Geometric Computing

Prof. Dr. Marcel Campen
Axel Schaffland, M.Sc.

UNIVERSITÄT OSNABRÜCK

# Computergrafik (SS 2018)

## Übung 8

fällig Montag **11. Juni**, 14:20

- Geben Sie bei jeder Abgabe alle Ihre Matrikelnummern auf jedem Blatt an.
- Verspätet eingereichte Abgaben können nicht gewertet werden.

**Aufgabe 1** (Theorie: Netze)

(a) (2P) In der Vorlesung haben wir mittels der Euler-Formel die durchschnittliche Valenz von Vertices in geschlossenen Dreiecksnetzen hergeleitet. Leiten Sie die durchschnittliche Vertex-Valenz eines geschlossenen *Hexagonnetzes* (ein Netz, welches aus sechseckigen Elementen besteht) mit Genus 1 her.

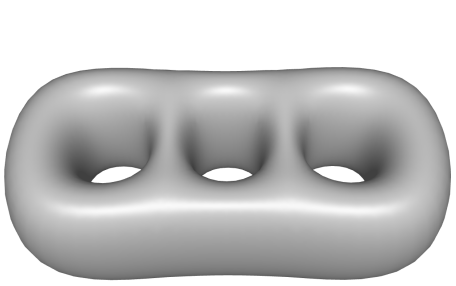
(b) (2P) Schreiben Sie (in beliebigem Pseudocode) einen möglichst einfachen Algorithmus, der zu einem gegebenen Vertex $v$ eines geschlossenen Dreiecksnetzes einen (beliebigen) anderen Vertex $w$ bestimmt, der nicht über eine Edge, jedoch über einen Pfad bestehend aus zwei Edges von $v$ aus erreicht werden kann. Nehmen Se dazu an, dass das Netz in der Halfedge Mesh Datenstruktur vorliegt, d.h. Sie können die Operationen `next(...)`, `opposite(...)`, `to(...)`, `face(...)`, `halfedge(...)`, und `out(...)` verwenden. Hinweis: 1-5 Zeilen Pseudocode sollten ausreichen.

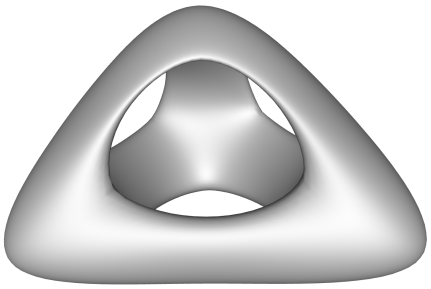
(c) (2P) Nehmen Sie an, dass ein Integerwert 4 Byte und eine Gleitkommazahl 8 Byte belegen. Nehmen Sie weiter an, dass Koordinaten als Gleitkommazahlen repräsentiert werden. Berechnen Sie wieviel Speicher benötigt wird, um das unten rechts abgebildete Dreiecksnetz in einer *Face List* Datenstruktur zu repräsentieren (wie sie beispielsweise im Rahmen des .STL-Dateiformates verwendet wird). Beachten Sie dabei ausschließlich den Speicher, der von den für das Netz relevanten Integerwerten (Indizes) und Gleitkommazahlen (Koordinaten) belegt wird, nicht etwaigen Overhead zur Listenverwaltung, Datei-Header oder Ähnliches.

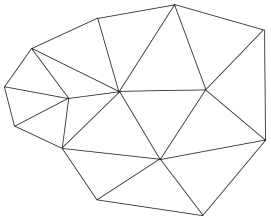
(d) (2P) Unter den gleichen Annahmen und Vorgaben wie in der vorigen Teilaufgabe: Berechnen Sie, wieviel Speicher zur Speicherung in einer *Shared Vertex* Datenstruktur (wie sie beispielsweise im Rahmen des .OFF-Dateiformates verwendet wird) benötigt wird.

(e) (2P) Unter den gleichen Annahmen und Vorgaben wie in der vorigen Teilaufgabe: Berechnen Sie, wieviel Speicher zur Repräsentation in einer *Halfedge Mesh* Datenstruktur (mit `next`-, `opposite`-, `to`-, `face`-, `halfedge`-, und `out`-Verlinkungen) benötigt wird.

(f) (2P) Welchen Genus hat das links abgebildete Objekt? Welchen Genus hat das in der Mitte abgebildete Objekt?







**Aufgabe 2** (Praxis: Netze & Glättung)

(a) (5P) Füllen Sie die Lücke in der Funktion `smoothMesh()` indem Sie einen Algorithmus implementieren, der ein Dreiecksnetz glättet. Das Netz liegt in einer Halfedge Mesh Datenstruktur vor. Vertices sind nummeriert von 0 bis `numVerts-1`. Die üblichen Operatoren sind verfügbar, d.h. für eine Halfedge mit Index `h`: `next[h]`, `to[h]`, `face[h]` und `opp[h]`. Beachten Sie die eckigen Klammern (da diese Operatoren hier einfach als Arrays implementiert sind – die die Indices der jeweils verlinkten Elemente enthalten). Für einen Vertex mit Index `v`: `out[v]`. Für jedes Face mit Index `f`: `halfedge[f]`.

Die Koordinaten des Vertex mit Index `v` erhalten Sie (als Typ `Point3D`) durch die Methode `getPoint(v)`. Neue Koordinaten können Sie setzen durch die Methode `setPoint(v, new Point3D(x,y,z))`.

Ihr Algorithmus soll über alle Vertices (also im Grunde über die Zahlen `v` von 0 bis `numVerts-1`) iterieren. Pro Vertex `v` soll dabei der Mittelpunkt seiner 1-Ring-Vertices berechnet werden, und `v` dann so verschoben werden (mittels `setPoint(v, ...)`), dass die neue Position auf der Hälfte der Strecke zwischen seiner bisherigen Position und dem berechnetem Mittelpunkt liegt.

(b) (3P) Ergänzen Sie Ihren Code aus Teil a derart, dass die Iteration alle Vertices unverändert lässt, die am Rand des Netzes liegen, d.h. die mit einer Rand-Halfedge (erkennbar an `face[h] == -1`) benachbart sind . Definieren Sie dazu innerhalb der Funktion `smoothMesh()` eine Hilfsfunktion `isBoundary(v)`. Implementieren Sie in dieser einen kurzen Algorithmus, der `true` zurückgibt, wenn der Vertex `v` am Rand des Netzes liegt, sonst `false`. Nutzen Sie diese Funktion dann in der obigen Iteration, um zu entscheiden, welche Vertices übersprungen werden sollen. Allerdings sollen Rand-Vertices nur übersprungen (d.h. festgelassen werden), wenn die Variable `fixBoundary` (welche Sie über die Checkbox unter dem Canvas umschalten können) den Wert `true` hat.