Media Informatics /
Graphics & Geometric Computing

Prof. Dr. Marcel Campen
Steffen Hinderink

UNIVERSITÄT OSNABRÜCK

Computergrafik (SoSe 2023)

Blatt 8

fällig Sonntag **18. Juni**, 24:00

Verspätet eingereichte Abgaben können nicht berücksichtigt 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 Faces besteht) mit Genus 1 her.

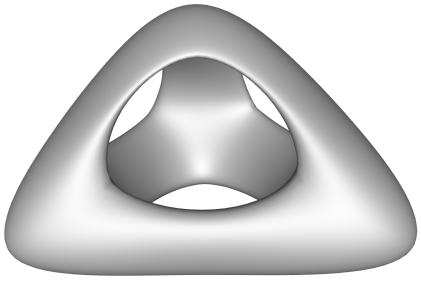
(b) (3P) Schreiben Sie (in beliebigem Pseudocode) einen möglichst einfachen Algorithmus, der zu einem gegebenen Vertex v eines geschlossenen Dreiecksnetzes einen (beliebigen) anderen Vertex w bestimmt, der nicht über eine Edge, jedoch über einen Pfad bestehend aus zwei Edges von v aus erreicht werden kann. Nehmen Sie an, dass das Netz in der Halfedge Mesh Datenstruktur vorliegt, d.h. Sie können die Operationen next, opposite, face, to, halfedge und out verwenden.

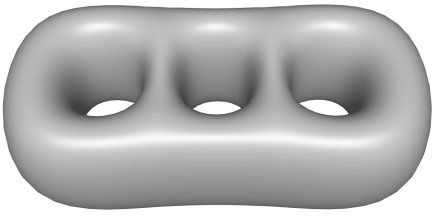
(c) (1P) Nehmen Sie an, dass ein Integerwert 4 Byte und eine Gleitkommazahl 8 Byte belegen. Nehmen Sie weiter an, dass Koordinaten als Gleitkommazahlen repräsentiert werden. Berechnen Sie wie viel Speicher benötigt wird, um das unten rechts abgebildete Dreiecksnetz in einer *Face List* Datenstruktur zu repräsentieren (wie sie beispielsweise im Rahmen des .STL-Dateiformates verwendet wird). Gehen Sie davon aus, dass das Dreiecksnetz dreidimensional ist. 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.

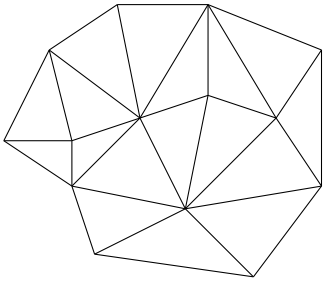
(d) (1P) Unter den gleichen Annahmen und Vorgaben wie in der vorigen Teilaufgabe: Berechnen Sie, wie viel Speicher zur Speicherung in einer *Shared Vertex* Datenstruktur (wie sie beispielsweise im Rahmen des .OFF-Dateiformates verwendet wird) benötigt wird.

(e) (1P) Unter den gleichen Annahmen und Vorgaben wie in der vorigen Teilaufgabe: Berechnen Sie, wie viel Speicher zur Repräsentation in einer *Halfedge Mesh* Datenstruktur (mit next-, opposite-, face-, to-, halfedge- und out-Verlinkungen) benötigt wird.

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







Media Informatics /
Graphics & Geometric Computing

Prof. Dr. Marcel Campen
Steffen Hinderink

UNIVERSITÄT OSNABRÜCK

Aufgabe 2 (Praxis: Netze & Glättung)

(a) (6P) Vervollständigen Sie die Funktion `smooth`, die ein Dreiecksnetz glättet. Das Netz liegt in einer Halfedge Mesh Datenstruktur vor. Die Vertices sind nummeriert von 0 bis `numVertices - 1`. Die üblichen Operatoren sind verfügbar, d.h. für eine Halfedge mit Index `h`: `next[h]`, `to[h]`, `face[h]` und `opposite[h]`. Beachten Sie die eckigen Klammern (da diese Operatoren hier einfach als Arrays implementiert sind, die die Indices der jeweils verlinkten Elemente enthalten). Für ein Face mit Index `f`: `halfedge[f]`. Für einen Vertex mit Index `v`: `out[v]`.

Die Koordinaten des Vertex mit Index `v` erhalten Sie (als Typ `Vector`) durch die Methode `getCoordinates(v)`. Neue Koordinaten können Sie setzen durch die Methode `setCoordinates(v, new Vector(x, y, z))`. Da die homogenen Koordinaten nicht mehr benötigt werden (da WebGL für das Rendern genutzt wird), wurden die Klassen `Vector` und `Point` zu `Vector` zusammengefügt.

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

Der Algorithmus soll bei einem Aufruf der Methode `smooth(n)` `n` mal ausgeführt werden. Die äußere Schleife ist dafür bereits implementiert. Mit den Buttons können Sie Ihre Funktion (mit 1 bzw. 100 Durchläufen) testen. Durch das Setzen der Checkbox Animate wird der Algorithmus in einer Endlosschleife immer wieder aufgerufen und das Netz dazwischen neu gerendert.

(b) (4P) Ergänzen Sie Ihren Code aus Teil a derart, dass alle Vertices unverändert bleiben, die am Rand des Netzes liegen. Ein Vertex liegt am Rand, wenn er inzident zu einer Rand-Halfedge (erkennbar an `face[h] === -1`) ist.

Definieren Sie dazu eine Funktion `isBoundary(v)`. Diese soll `true` zurückgeben, wenn der Vertex `v` am Rand des Netzes liegt, und sonst `false`. Nutzen Sie diese Funktion dann im Code von Teil a. Allerdings sollen Rand-Vertices nur übersprungen (d. h. festgelassen werden), wenn die Variable `fixBoundary` (welche Sie über die Checkbox umschalten können) den Wert `true` hat.

Aufgabe 3 (Bonus: Low Distortion Mapping)

Bestimmen Sie durch Low Distortion Mapping Texturkoordinaten für das Netz aus Aufgabe 2 und texturieren Sie das Netz (mit einer selbst gewählten Textur) mittels dieser Texturkoordinaten.

Sie können dazu einen sehr ähnlichen Algorithmus verwenden wie in Aufgabe 2 – allerdings angewendet auf 2D Texturkoordinaten statt auf 3D Vertex-Raumkoordinaten. Fixieren Sie dabei in der Funktion `resetTexture` die Randvertextexturkoordinaten auf einen Kreis (der Rand muss wegen des Satzes von Tutte konvex sein) und initialisieren Sie die übrigen mit $(0.5, 0.5)$. Verschieben Sie die Texturkoordinaten (analog zu den Vertexpositionen in der Funktion `smooth`) in der Funktion `spread`.