3D-Computergrafik I

Sommersemester 2015 – Aufgabenblatt 6

Hasso-Plattner-Institut
Fachgebiet Computergrafische Systeme

Aufgabe 19: Polygon Tessellation mit gluTesselator (7 Punkte)

Ziel dieser Aufgabe ist es, mit Hilfe des `GLUtesselators` eine Tessellation von beliebigen Polygonen umzusetzen (siehe Abbildung 1). Im gegebenen Programmrahmen können Punkte durch einen Linksklick auf das Canvas hinzugefügt werden. Diese Punkte werden als Eckpunkte der Kontur eines Polygons betrachtet. Mit einem Rechtsklick wird die aktuelle Kontur abgeschlossen.

Jede abgeschlossene Kontur wird in der Struktur `Contour` abgelegt, welche aus einer Menge von Punkten besteht. Alle aktuell abgeschlossenen Konturen befinden sich in der Liste `ContourList`.

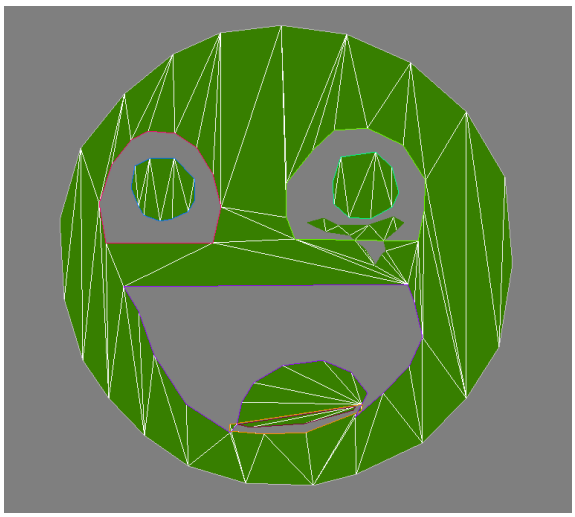


Abbildung 1: Beispiel einer Tessellation.

Zeichnen Sie alle Konturen aus dieser Liste, indem Sie in der Klasse `Exercise19` die Tessellation und das Rendering umsetzen. Implementieren Sie dazu die Funktion `tessellatePolygons`, welche die Konturen an den `GLUtesselator` übergibt. Hinweis: Es soll nur ein Tessellator Objekt benutzt werden.

Der Tessellator gibt nun die verarbeiteten Konturen durch Callbacks zurück. Zeichnen Sie diese Konturen, indem Sie die folgenden Callbacks implementieren:

beginCallback wird aufgerufen, wenn ein neues Primitiv begonnen wird.

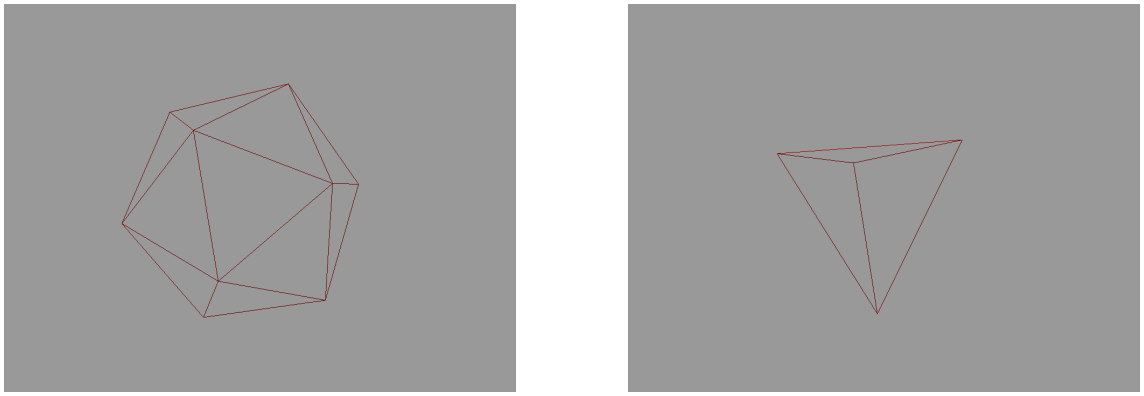
vertexCallback wird für jeden vom Tessellator bearbeiteten Punkt aufgerufen.

endCallback wird am Ende der aktuellen Kontur aufgerufen. Zeichnen Sie hier die aktuelle Kontur einerseits gefüllt und andererseits als Linienzug.

Weitere Informationen im Moodle und unter `http://www.glprogramming.com/red/chapter11.html`.

Aufgabe 20: Polyeder Tessellation mit CGAL (14 Punkte)

Ziel dieser Aufgabe ist es, mit CGAL Polyeder zu definieren, zu verfeinern und für OpenGL aufzubereiten. Verwenden Sie dabei beispielhaft sowohl einen regulären Ikosaeder als auch einen regulären Tetraeder (Abbildung 2). Ein Beispiel für eine Verfeinerung eines regulären Ikosaeders zeigt Abbildung 3. Die Tasten `w`, `s` und `p` schalten jeweils den Renderingmodus (Wireframe/Normal), die Verfeinerungsmethode und den verwendeten Polyeder um. Außerdem ist ein Slider in der GUI enthalten, mit dem der Verfeinerungsgrad eingestellt werden kann. Gehen Sie bei der Implementierung in `exercise20.cpp` wie folgt vor:



(a) (b)

Abbildung 2: Regulärer Ikosaeder (a) und Tetraeder (b) als Wireframe dargestellt.

a) Definieren Implementieren Sie in der Methode `createMesh()` die Erstellung des regulären Ikosaeders und des regulären Tetraeders. Verwenden Sie dazu die Variable `builder` vom Typ `CGAL::Polyhedron_incremental_builder_3<>`. Die Membervariable `m_polyhedronMode` legt fest, welcher der beiden Polyeder erstellt wird. Außerdem sollen die Polyeder im Ursprung zentriert sein.

b) Verfeinern Implementieren Sie in der Methode `prepareMesh(Polyhedron& poly)` zwei verschiedene Verfeinerungsmethoden für Dreiecksnetze. Verwenden Sie dazu Verfeinerungsmethoden aus dem Namespace `CGAL::Subdivision_method_3`, beispielsweise `Loop_subdivision(...)` und `Sqrt3_subdivision(...)`. Die für die Verfeinerungsmethoden verwendete Anzahl an Iterationen ($[0, 8] \in \mathbb{N}$) soll in Abhängigkeit von der Membervariable `m_animationFrame` ($[0, 1] \in \mathbb{R}$) erfolgen.

c) Aufbereiten Implementieren Sie in der Methode `prepareMesh(Polyhedron& poly)` die Initialisierung der Vertex- und Normalenarrays für OpenGL. Iterieren Sie dazu über die Half-Edge-Struktur des Funktionsparameters `poly`. In der Half-Edge-Struktur sind die Normalen pro Face definiert und sollen für alle Vertices des Face gelten. Für eine Annäherung an eine Kugel soll jeder Vertex als Richtungsvektor interpretiert werden und der darzustellende Vertex soll aufgrund des Richtungsvektors und einer festen Länge ermittelt werden (z.B. 2).

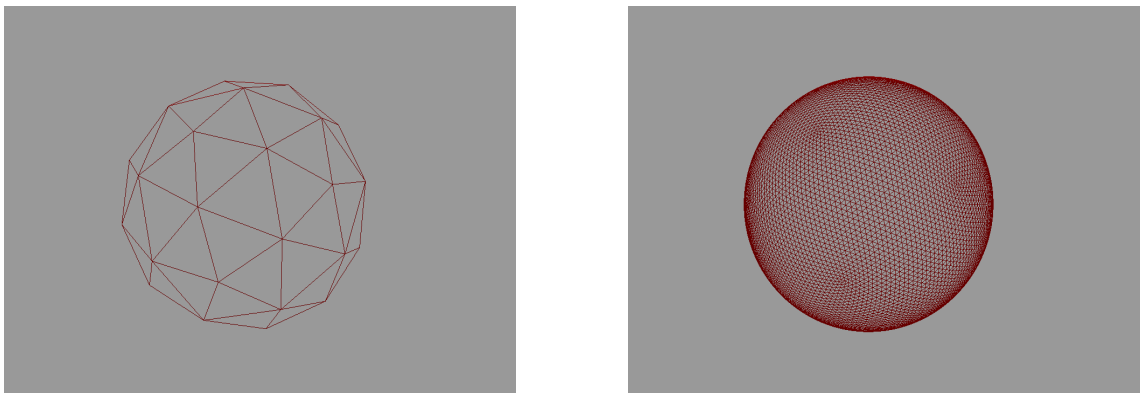


Abbildung 3: Verschiedene Verfeinerungsgrade eines regulären Ikosaeders als Wireframe dargestellt.

Vorbereiten von CGAL:

- CGAL ≥ 4.5 wird benötigt
- Zusätzliche, externe Abhängigkeiten sind Boost und GMP
  - getestet für Boost $\geq 1.55.0$
  - Linux/Mac: Empfohlen via Paketverwaltungssystem (`libboost-all-dev` und `libgmp-dev`).
  - Windows: Empfohlen via Installer (`http://sourceforge.net/projects/boost/files/boost-binaries/`).
  - Es werden dynamische Boost Bibliotheken empfohlen.
- Installation
  - Linux/Mac: Paketverwaltungssystem (`libcgal-dev`) oder mit CMake Quellcode (`https://gforge.inria.fr/frs/?group_id=52`) selber kompilieren.
  - Windows: Mit CMake Quellcode (`https://gforge.inria.fr/frs/?group_id=52`) selber kompilieren.
  - Hinweis zu der CGAL CMakeLists: Von den `WITH_*` Optionen reichen `WITH_CGAL_Core` und `WITH_GMP` aus.

Allgemeine Hinweise:

- Die Aufgaben sollen maximal zu zweit bearbeitet werden; Ausnahmen müssen mit den Übungsleitern abgesprochen werden.
- Bitte reichen Sie Ihre Lösungen bis Dienstag, den **07.07.2015**, um **13.15** Uhr ein.
- Tragen Sie Ihre Matrikelnummern in die Quellcode-Dateien ein. Beachten Sie, dass nur die vollständigen Quelltexte und Projektdateien geschickt werden sollen. Senden Sie uns keinesfalls ausführbare Dateien oder bereits kompilierte Binär- oder Temporärdateien (*.obj, *.pdb, *.ilk, *.ncb etc.) zu! Testen Sie vor dem Verschicken, ob die Projekte aus den Zip-Dateien fehlerfrei kompiliert und ausgeführt werden können.
- Zippen Sie Ihre Lösungen in **eine** Zip-Datei. Geben Sie der Zip-Datei einen Namen nach folgendem Schema:

  cg1_blatt6_matrikelnummer1_matrikelnummer2.zip.
- Die Zip-Datei laden Sie dann bei moodle hoch.