Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. Dr. M. Gross
Remo Ziegler / Christian Voegeli / Daniel Cotting

Abgabe: 18. April 2006

Visual Computing
OpenGL Rendering

Ziele

- Kennenlernen der Übungsumgebung, inkl. *OpenGL* und dem *OpenGL Utility Toolkit (GLUT)*.
- Kennenlernen des *OpenGL*-Konzeptes der Display-Listen.
- Kennenlernen der Färbungs- und Schattierungsmöglichkeiten von Objekten in *OpenGL*.
- Verständnis der Anwendung von Farbräumen zur Visualisierung wissenschaftlicher Daten.

Allgemeine Hinweise

Alle praktischen Übungsmodule werden auf PCs unter Windows durchgeführt. Als Programmierumgebung wird Microsoft Visual Studio .NET 2003 eingesetzt. Die gesamte Hilfe dazu ist über MSDN oder online unter `http://msdn.microsoft.com/` verfügbar.

Im Weiteren werden die OpenGL sowie die GLUT Libraries verwendet. Auf der Webseite der Vorlesung finden sich unter *Course notes* Links zu den für die Übungen benötigten, offiziellen Hilfeseiten. Unter *Exercises* sind neben weiteren Informationen auch die einzelnen Übungsmodule verfügbar.

Für die Benutzung vorbereitet ist das PC Lab im IFW im Raum C31. Die Übungsabgabe erfolgt für alle praktischen Übungen per E-mail an den jeweils betreuenden Assistenten.

Ressourcen

Webseite der Vorlesung: `http://graphics.ethz.ch`, dann Link *Courses Overview* und Link *Visual Computing*.

1) Rendern eines Oberflächen-Modells

Das erste Übungsmodul kann von der Webseite der Vorlesung heruntergeladen und nach seiner Dekomprimierung in ein neues Verzeichnis als Visual Studio .NET 2003 Solution geöffnet werden.

In dieser Aufgabe soll ein beliebiges, dreiecksbasiertes Objekt in OpenGL gerendert werden. Als Beispielsdatensatz werden wir zwei Oberflächen-Modelle des Vulkanes "Mount St. Helens" (`http://volcano.und.edu/vwdocs/msh`) verwenden, wobei der Datensatz *vulcano_mesh1.raw* das Relief des Vulkanes vor und *vulcano_mesh2.raw* nach dessen Ausbruch darstellt.

Im vorliegenden Programm wird aus einem File die Oberflächenbeschreibung eines Objektes eingelesen und in lokalen Variablen gespeichert. Es werden zwei Listen erstellt. Die eine Liste enthält eine Anzahl (`nbr_of_nodes`) von Knoten mit ihren 3D Koordinaten. Die andere besteht aus einer Anzahl (`nbr_of_triangles`) von Dreiecken, welche durch ihre Eckpunkte als Referenzen in die Knotenliste gegeben sind.

a) Displayliste

Erstellen Sie aufgrund dieser Datenstruktur eine Display-Liste in OpenGL, mit welcher das Objekt dargestellt werden kann. Kreieren Sie dazu am Ende der Funktion *loadObject()* eine aus einzelnen Dreiecken bestehende Display-Liste.

Benutzen Sie dabei die OpenGL Funktionen

- `glNewList(...)` und `glEndList()` zur Definition einer (kompilierten) Display-Liste
- `glBegin(...)` und `glEnd()` zur Markierung des Beginns und des Endes der Dreiecksdefinition
- `glVertex3fv(...)` zur Definition der Eckpunkte

Traversieren Sie die Liste der Dreiecke `triangle_array` und holen Sie sich die Positionsdaten der Dreieckseckpunkte anhand der Referenzen in die Liste der Knoten `node_array`. Auf die erste Koordinate des ersten Eckpunktes des Dreieckes mit dem Index `triangle_index` lässt sich z.B. mit `node_array[triangle_array[triangle_index].node_index_A][0]` zugreifen.

Um die Display-Liste zu rendern, muss sie in der Display-Callback-Funktion *display()* an der dafür vorgesehenen Stelle mit

- `glCallList(...)`

aufgerufen werden.

Als Ergebnis dieser Teilaufgabe sollten Sie eine gleichmässig grau gefärbte Fläche zu sehen bekommen. Um eine Schattierung der einzelnen Dreiecke berechnen zu können, werden nun noch Oberflächennormalen benötigt. Diese können entweder per Dreiecksfläche oder per Eckpunkt berechnet werden.

b) Flächennormalen

Es soll für jedes Dreieck ein Vektor berechnet werden, welcher senkrecht zur Dreiecksfläche steht. Traversieren Sie dazu, noch vor der Generierung der Display-Liste, die Liste der Dreiecke `triangle_array`. Berechnen Sie für jedes Dreieck die Flächennormale und speichern Sie diese in dem Vektor `triangle_array[triangle_index].face_normal` ab.

Setzen Sie nun innerhalb der Definition der Display-Liste mit der Funktion

- `glNormal3fv(...)`

für jede Dreiecksfläche, die korrekte Flächennormale.

c) Eckpunktnormalen

Anstatt nur eine Normale pro Dreiecksfläche zu berechnen, kann man auch in jedem Knotenpunkt eine korrekte Oberflächennormale speichern. Dies ermöglicht es, die Helligkeitswerte, welche in den Dreieckseckpunkten berechnet werden, über die Dreiecksfläche zu interpolieren. Die Oberflächennormale in einem Knotenpunkt berechnet sich durch eine Mittelung der zuvor berechneten Flächennormalen aller an den Knoten angrenzenden Dreiecksflächen.

Rufen Sie in der Funktion *loadObject()* vor der Definition jedes Eckpunktes die Funktion *setVertexNormal(...)* auf, in welcher Sie für den angegebenen Knoten die korrekte Oberflächennormale berechnen und ebenfalls mit `glNormal3fv(...)` definieren.

2) Umwandlung HSV-Farbraum nach RGB-Farbraum

Zur Einfärbung des in der vorhergehenden Teilaufgabe erstellten Oberflächen-Modelles, wollen wir uns des HSV-Farbraumes bedienen. Die Definition von Farben in OpenGL erfolgt aber über RGB-Farbkoordinaten. Aus diesem Grund brauchen wir eine Umwandlungsroutine von HSV nach RGB. Implementieren Sie die Funktion *hsv2rgb(h, s, v, &r, &g, &b)*, welche einen im HSV-Farbraum gegebenen Farbpunkt in den RGB-Farbraum transformiert.

3) Einfärben des Oberflächen-Modelles

Bei der Visualisierung von wissenschaftlichen Daten steht man oft vor der Schwierigkeit, gleichzeitig möglichst viele Eigenschaften eines Objektes möglichst übersichtlich in einem Bild darzustellen. In dieser Aufgabe sollen zwei Eigenschaften durch geeignete Farbgebung visualisiert werden.

a) Höhenkarte

Zum einen soll die Höhe des Reliefs durch eine Farbtabelle widerspiegelt werden, welche nur reine Farben des Farbkreises beinhaltet.

Rufen Sie dazu in *loadObject(...)* vor jeder Definition eines Eckpunktes zusätzlich noch die Funktion *setVertexColor(...)* auf, und verwenden Sie dann die in der vorhergehenden Aufgabe implementierte Funktion *hsv2rgb(h, s, v, &r, &g, &b)* zur Implementation von *setVertexColor(...)*. Benutzen Sie zum Setzen der Farbe die OpenGL Funktion

- `glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, color)`, wobei `color` ein vier dimensionaler Vektor ist, welcher in den ersten drei Koordinaten den Farbwert in RGB und als vierten Wert die Transparenz (alpha channel) enthält. (Der Transparenzwert wird in dieser Aufgabe auf 1 gesetzt, was einem soliden Körper entspricht.)

b) Oberflächeneigenschaft

Im weiteren soll eine bestimmte Oberflächeneigenschaft speziell hervorgehoben werden, ohne aber die Darstellung der Höheninformation zu verlieren. Dazu nehmen wir beispielsweise an, dass an einem bestimmten Punkt (`interesting_point`) eine uns speziell interessierende Gesteinsart auftritt, deren Anteil an der Gesamtoberfläche mit der Entfernung zu diesem Punkt linear abnimmt. Um diese Eigenschaft zu visualisieren, soll, ausgehend von der Fläche, die kein solches Gestein enthält, immer mehr Weiss zu der bestehenden Farbe gemischt werden, je näher man zum Zentrum der Verteilung der uns interessierenden Gesteinsart kommt.