Praktikum Computergraphik

– Übungsblatt 1 –

Lehrstuhl für Computergraphik und Multimediasysteme

Dmitri Presnov, Markus Kluge

Überblick

In dieser Aufgabe soll ein ein Tychonisches Weltmodell modelliert werden, welches um einen Todesstern erweitert wird.

Den Mittelpunkt des geozentrischen Systems bildet die Erde, die von Sonne und Mond umkreist wird. Die als Lichtquelle fungierende Sonne wird von weiteren Planeten umkreist, welche wiederum von zusätzlichen Monden umkreist werden können.

Alle Planeten und Monde bewegen sich auf Kreisbahnen mit unterschiedlichem Radius und unterschiedlichen Geschwindigkeiten und haben zusätzlich eine lokale Rotation um die eigene Achse.

Um einen der Planeten schwebt ein Todesstern, der mit seinem Laser andere Planeten anleuchten kann.

Über die grafische Benutzerschnittstelle des Programms können diverse Parameter des Systems, der Beleuchtung sowie des Renderns eingestellt werden.

Themenschwerpunkte

- Grundlagen der OpenGL- und Qt-Programmierung
- Algorithmische Generierung von 3D-Modellen
- Beleuchtung und Texturierung
- Hierarchischer Szenen-Aufbau
- Animationen

Allgemeines

Achten Sie auf sauber strukturierten und gut dokumentierten Code. Ziel dieses Praktikums ist es nicht nur, die Grafikprogrammierung zu erlernen, sondern auch größere C++ Projekte umzusetzen. Codequalität ist hier wichtiger als eine besonders ästhetische Wahl der Renderparameter.

Sie haben beim Bearbeiten der Aufgaben gewisse Freiheiten. Kreative Lösungen oder eigenständige Erweiterungen sind gerne gesehen. Dennoch sollten alle geforderten Ziele umgesetzt werden, sprechen Sie also sicherheitshalber ihre Ideen mit den Betreuern ab, bevor Sie diese umsetzen.

Sollte eine Aufgabenstellung unklar sein, sprechen Sie mit den Betreuern. Dort können Sie auch die fertig implementierte Musterlösung einsehen (das Programm, nicht den Quellcode), um Fragen zu klären.

In vielen Schritten sind einige wichtige Funktionen angegeben, die Sie mit hoher Wahrscheinlichkeit für das Lösen dieses Schrittes benutzen müssen. Es empfiehlt sich daher, vor Bearbeitung jedes Schrittes diese Funktionen zu studieren und erst mit der Aufgabe zu beginnen, wenn ihre Verwendung verstanden wurde.

GLM

Dieses Programm verwendet zusätzlich zu OpenGL und Qt die Bibliothek GLM. GLM bietet einfach zu benutzende Vektoren- und Matrizenklassen nach dem Vorbild der OpenGL Shading Language. Das Projekt ist so konfiguriert, dass GLM ohne weiteres Einrichten direkt verwendet werden kann.

Da in diesem Programm die Matrizenfunktionen von OpenGL benutzt werden, werden nur die Vektorenklassen von GLM benötigt.

Je nach Größe des Vektors gibt es die 3 Klassen vec2, vec3 und vec4. Diese bieten die üblichen Operatoren für Addition und Skalarmultiplikation sowie direkten Zugriff auf die bis zu 4 Komponenten per 'x, y, z, w' oder auch 'r, g, b, a' für Farbwerte. Darüber hinaus gibt es nützliche Funktionen für die Länge von Vektoren oder das Kreuz- und Skalarprodukt. Zur Dokumentation von GLM kann man sich entweder die GLSL-Spezifikation oder die GLM-Dokumentation, ansehen.

1. Einleitung und Überblick

Machen Sie sich mit dem Grundgerüst des Programms vertraut. Benutzen Sie dafür CMake um für die Entwicklungsumgebung ihrer Wahl eine Projekt- bzw. Make-Datei zu erstellen. Kompilieren Sie das Grundgerüst und starten Sie testweise die Anwendung. Es sollte ein rotierender, bunter Würfel zu sehen sein.

Als nächstes sollten Sie sich die einzelnen Klassen ansehen. Es sind bereits die wichtigsten Klassen angelegt, die Sie zur Bearbeitung der Aufgaben benötigen, allerdings fehlen ihnen größtenteils erhebliche Teile der Implementierung, die Sie selber vervollständigen sollen. Sie werden dabei nicht nur Methoden fertig implementieren müssen, sondern auch neue Methoden und Attribute deklarieren müssen. Wenn Sie zusätzlich weitere Klassen anlegen möchten, ist dies auch möglich, dann muss jedoch auch die CMake-Datei erweitert werden.

2. OpenGL

Das Projekt soll sich an der OpenGL Shading Language 4.0 orientieren, welche sich nicht an der Fixed Function Pipeline orientiert. Der vorgegebene Programmcode zeigt den konkreten Einsatz am Beispiel eines bunten Würfels. Bevor sie beginnen zu programmieren, sollten sie das Minimalbeispiel verstanden haben. Sollten Sie damit größere Schwierigkeiten haben, empfiehlt es sich ein Einsteigertutorial im Internet zu suchen um einen ersten Einblick zu bekommen.

3. Shader

Die Shader-Dateien sind über das Qt Ressource System eingebunden.

Wenn Sie einen neuen Shader erstellen, sollten Sie ihn zunächst in der CMakeLists.txt hinzufügen, damit er im Projektbaum angezeigt wird. Dann müssen Sie ihn in der Datei shader.qrc hinzufügen, um ihn benutzen zu können.

Wie Sie auf den Shader zugreifen können, entnehmen Sie dann der Funktion *Planet::getVertexShader* bzw. der Funktion *Drawable::loadShaderFile*.

4. Kamerabewegung

Die Bewegung der Kamera ist nicht nur wichtig, um später die Szene von allen Seiten betrachten zu können, sie ist auch ein unverzichtbares Werkzeug zum Debuggen ihrer Anwendung. Viele Renderfehler bemerkt man erst, wenn man die Szene von einer anderen Seite betrachtet.

In dieser Aufgabe soll der Benutzer die Kamera mit Hilfe der Maus um das System drehen können. Die Kamera bewegt sich dabei auf einer Kugel um die Szene herum und schaut immer auf den Mittelpunkt.

Es bietet sich an, die Kameraposition in Polarkoordinaten zu speichern, wobei je ein Winkel von einer Mausachse manipuliert wird. Der Abstand zur Erde sollte durch das Mausrad beeinflusst werden können. Die dafür nötigen Event-Handler sind als Grundgerüst bereits in der `GLWidget`-Klasse vorhanden.

Wichtige Funktionen: `sin()`, `cos()`, `gluLookAt() b.z.w. glm::lookAt()`

5. SkyBox

Für den Hintergrund der Szene soll eine Skybox verwendet werden. Recherchieren Sie zunächst die Funktionsweise einer Skybox und vervollständigen Sie dann die entsprechende Klasse.

Es bietet sich an, beim Rendern der Skybox den Depth-Buffer zu deaktivieren, auf diese Weise muss die Skybox keine riesigen Ausmaße haben, um sicher zu gehen, dass später keine Teile der Szene von ihr verdeckt werden können.

Die Skybox besteht aus 6 Vierecken[1], von der jedes eine komplette Textur zeigt. Beim Rendern ist es äußerst wichtig, darauf zu achten, dass sich die Kamera stets im Mittelpunkt der Skybox befindet, da ansonsten die Illusion zerstört wird.

Wichtige Funktionen: `glDisable()`, `glBindTexture()`, `glTexImage2D()`

Wichtige Begriffe: `GL_TEXTURE_CUBE_MAP`, `GL_TEXTURE_CUBE_MAP_POSITIVE_X`, `GL_DEPTH_TEST`

6. Geometrische Grundkörper

Das gesamte System wird nur aus 3 verschiedenen Objekttypen zusammengebaut - Kugeln, Ringe und Kegeln. Da diese häufig vorkommen, sollten sie jeweils in eine Klasse gekapselt werden.

Überlegen Sie sich jeweils eine geeignete Parametrisierung für die Oberfläche einer Kugel, eines Rings und eines Kegels. Dabei ist zu beachten, dass durch Polygone runde Oberflächen immer nur angenähert werden können und ihre Darstellung nur eine endliche Auflösung haben kann. Zur Optimierung der Performance sollte diese Auflösung pro Objekt angepasst werden können, um kleine Objekte mit niedriger und große Objekte mit hoher Auflösung (Polygonauflösung, nicht Pixelauflösung) rendern zu können.

Außerdem benötigen die Modelle später Texturkoordinaten und Normalvektoren für die Beleuchtung.

[1] Es ist auch möglich eine Skybox mit nur einem Viereck zu erzeugen. Das ist aber hier nicht nötig.

Um die Modellierung zu verifizieren, soll es möglich sein über die Benutzeroberflächen den Polygon Modus zu aktivieren, in dem alle Objekte nur als Gitternetzmodelle gezeichnet werden.

7. Das System

Mithilfe der Kugeln wird nun das System modelliert. Dabei bildet die Erde den Mittelpunkt und wird von Sonne und Mond umkreist. Um die Sonne kreisen die Planeten Merkur, Venus, Mars, Jupiter und Saturn. Zusätzlich ist der Jupiter von seinen Monden Io, Europa, Ganymed und Callisto umgeben, während um den Mars ein Todesstern kreist.[2]

Der Saturn ist von einem Ring umgeben. Dieser soll so animiert werden, dass das Innere des Rings tatsächlich l̈eerïst. Es soll also keine Scheibe modelliert werden.

Die Auflösung[3], mit der die einzelnen Modelle gerendert werden, soll über die Benutzeroberfläche konfigurierbar sein.

Optional sollen die Umlaufbahn der Planeten angezeigt werden können. Einerseits sollen die *lokalen* Umlaufbahnen gezeichnet werden. Diese zeigen lediglich die Bahn des Planeten relativ zu dem ihm übergeordneten Planeten an. Die lokalen Bahnen bilden also perfekte[4] Kreise.

Alternativ soll es auch möglich sein sich die *globale* Flugbahn der einzelnen Planeten anzeigen zu lassen. Diese ist immer relativ zur Erde gemessen und ändert sich nicht, wenn sich die Planeten weiterbewegen. Da sich die Bahnen nicht verändern, sollen sie nur einmal beim Start des Programms einmal berechnet werden, um die Performance zu steigern. Ein Grundgerüst zur Berechnung der Bahnen ist bereits vorgegeben.

Wichtige Funktionen: `glGenVertexArrays()`, `glBindVertexArray()`, `glGenBuffers()`, `glBindBuffer()`, `glBufferData()`, `glVertexAttribPointer()`, `glDeleteBuffers()`, `glUseProgram()`, `glGetUniformLocation()`, `glUniformMatrix4fv()`, `glUniform1i()`, `glUniform3f()`, `glActiveTexture()`, `glBindTexture()`, `glDrawElements()`,

8. Texturierung

Die benötigten Texturen sind bereits in das Programmgrundgerüst integriert. Bei einem erfolgreichen Durchlauf von CMake werden die nötigen Bilddateien in das *build* Verzeichnis kopiert und können somit mittels relativer Pfade angesprochen werden.

Bei der Texturierung der Erde sollen zwei Texturen so übereinander gelegt werden, dass der Eindruck entsteht, dass Wolken über die Erde ziehen. Dies soll mittels einer Überlagerung im Shader geschehen. Es ist nicht vorgesehen, dass für die Erde zwei unterschiedlich große Kugeln modelliert werden.

9. Beleuchtung

Nun soll die Szene beleuchtet werden. Vorgesehen ist eine Punktlichtquelle, welche von der Sonne ausgeht und eine Punktstrahler, der sich auf dem Todesstern befindet.

Die Sonne beleuchtet die Planeten so, dass immer eine Seite hell und die andere Seite dunkel ist. Der Einfachheit halber werfen die Planeten keine Schatten aufeinander.

---

[2] Die Hierarchie inklusiver aller Radien, Geschwindigkeiten und Abstände ist im Programmcode bereits vorgegeben.

[3] Hier: Die Anzahl der Dreiecke

[4] theoretisch

Auf dem Todesstern befindet sich ein *Laser*, der sich zusammen mit dem Todesstern dreht und einen roten *Spot* auf angeleuchtete Planeten wirft. Der *Cutoff* Winkel soll über die Benutzeroberfläche steuerbar sein.

10. Animation

Im letzten Schritt sollen die Körper animiert werden. Es gibt dabei 2 Bewegungen: Die Eigenrotation der Planeten und die Drehung der Planeten untereinander.

Durch einen bereits implementierten Timer wird die `GLWidget::animateGL`-Methode in regelmäßigen Abständen aufgerufen. Diese Funktion sollte daher alle zeitlichen Abläufe steuern. Die Benutzeroberfläche sollte um einen Regler erweitert werden, mit dem die Geschwindigkeit der Animation verstellt werden kann um die Flugbahn der Planeten verifizieren zu können oder die Simulation ganz zu stoppen.

Die Animationen werden dabei durch die Transformationsmatrizen umgesetzt.

Bewertung

Bei der Bewertung werden folgende Kriterien berücksichtigt:

1. Die Aufgaben sind gemäß der Aufgabenstellung gelöst.
2. Das gesamte Projekt unterliegt einer objektorientierten Denkweise, genügt den Grundlagen der modularen Gestaltung und ist übersichtlich editiert.
3. Der Quellcode ist in englischer Sprache kommentiert und enthält sämtliche Hilfen, die zum Verständnis der Programmstruktur bzw. der verwendeten Algorithmen notwendig sind.
4. In der mündlichen Abnahme sollte jeder Teilnehmer in der Lage sein, den Programmcode kurz zu erläutern und allgemeine Fragen zu OpenGL beantworten zu können.