

Praktikum Computergraphik

– Übungsblatt 1 –

Lehrstuhl für Computergraphik
und Multimediasysteme

Christoph Schikora, David Bulczak

Überblick

In dieser Aufgabe soll ein ein Doppelsternsystem mit einem Dyson Schwarm modelliert werden.

Die als Lichtquelle fungierende Sonne kreist zusammen mit einem Neutronenstern, als Doppelsystem um den Mittelpunkt der Szene. Diese bewegen sich auf Kreisbahnen mit unterschiedlichem Radius, halten jedoch die gegenüberliegende Position zueinander.

Ebenfalls auf Kreisbahnen befinden sich Habitate eines Dyson Schwarms. Ein spezielles Habitat auf der äußeren Schale soll ebenfalls Licht als Punktstrahler in Richtung des Zentrums abgeben.

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 Koordinatensystem zu sehen sein.

Als nächstes sollten Sie sich die einzelnen Klassen ansehen. Es sind bereits alle 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. 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 Mausechse manipuliert wird. Die Größe der Kugel sollte durch das Mousrad beeinflusst werden können. Die dafür nötigen Event-Handler sind als Grundgerüst bereits in der `RenderWindow`-Klasse vorhanden.

Wichtige Funktionen: `sin()`, `cos()`, `gluLookAt()`, `glMatrixMode()`

3. 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, von der jedes eine komplette Textur zeigt. Die benötigten Texturen werden bereits vom Programmgrundgerüst geladen. 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: `glDepthMask()`, `glBindTexture()`, `glBegin()`, `glEnd()`, `glTexCoord2f()`, `glVertex3f()`, `glTranslatef()`

4. Geometrische Grundkörper

Das gesamte System wird nur aus 2 verschiedenen Objekttypen zusammgebaut - Kugeln und Zylinder. Da diese häufig vorkommen, sollten sie jeweils in eine Klasse gekapselt werden.

Überlegen Sie sich jeweils eine geeignete Parametrisierung für die Oberfläche einer Kugel und eines Zylinders. 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.

5. Das System

Mithilfe der Kugeln wird nun das Zentrum des Systems modelliert. Dabei sollen sich Sonne und Neutronenstern auf 2 Kreisbahnen mit unterschiedlichem Radius an den jeweils entgegengesetzten Enden bewegen.

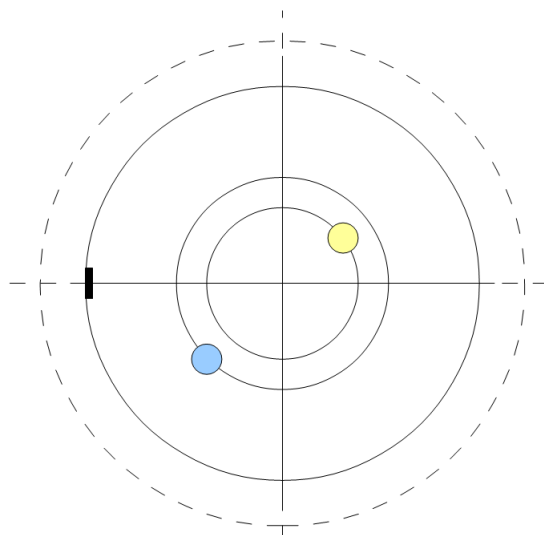


Abbildung 1: Aufbau des Systems

Die Zylinder dienen der Modellierung der Habitate. Diese sollen sich auf Kreisbahnen um das Zentrum des Systems bewegen. Dabei sind mehrere Schalen vorgesehen. Es soll angestrebt werden die Bahnen möglichst unterschiedlich zu platzieren. Auf der ersten Schale sollen 3 Bahnen entstehen, nämlich in den Ebenen XY, YZ und XZ. Auf den nächsten Schalen sollten Sie

die Bahnen jeweils anders platzieren, wobei die Anzahl an Bahnen mit jeder Schale zunimmt. Die Bahnen auf den folgenden Schalen sollen, jeweils relativ zur vorherigen Schale, möglichst zwischen den Bahnen der vorherigen Schale liegen. Innerhalb der Schalen wird keine Kollisionsvermeidung zwischen den Habitaten erwartet, lediglich bei einem Habitat pro Bahn wird erwartet, dass die Bewegung/Position der Habitate so gewählt ist, dass diese nicht kollidieren. Wählen Sie zumindest für diesen Fall eine möglichst gleichmässige Verteilung. Die Habitate sollen entlang der Binormalen der Bahn rotieren.

Es soll wahlweise die Anzahl der Habitate oder die Anzahl der (vollen) Schalen über die Benutzeroberfläche eingestellt werden können (achten Sie dabei auf sinnvolle Minimal- und Maximalwerte). Außerdem soll festgelegt werden können, wie viele Habitate sich auf einer Kreisbahn befinden.

Optional soll auch die Umlaufbahn der Habitate angezeigt werden können. Diese Funktion ist vor allem praktisch um zu testen, ob das hinzufügen neuer Schalen funktioniert.

Um die Szene aufzubauen sollte der Matrix-Stack von OpenGL benutzt werden.

Wichtige Funktionen: `glLoadIdentity()`, `glPushMatrix()`, `glPopMatrix()`, `glRotatef()`, `glTranslatef()`

6. Texturierung

Die benötigten Texturen sind bereits in das Programmgrundgerüst integriert. Zu beachten ist, dass das Qt-Ressourcensystem benutzt wird, die Texturen werden also während dem Compilieren in die ausführbare Datei gepackt und auch von dort geladen, so dass das Programm von den ursprünglichen Dateien unabhängig ist. Die Ressourcen-Definition steht dabei in der 'qtmidi.qrc', die mithilfe des Qt-Designers oder einem beliebigen Texteditors editiert werden kann.

Ein Beispiel für die Benutzung bietet schon das Grundgerüst, der Skybox, welches die Texturen aus der Ressourcendatei lädt. Nähere Informationen zu dem Ressourcensystem befinden sich in der Qt-Dokumentation.

Die Sonne und die Habitate erhalten hier einfache Texturen. Es ist zu beachten, dass das Habitat, welches später als Lichtquelle dient, eine andere Textur erhält.

Der Neutronenstern soll aus 3 Kugeln bestehen, wobei die Innere sich von den Äußeren unterscheidet, welche zusätzliche Alphawerte haben, damit auch die inneren Schalen sichtbar sind. Die Kugeln/Texturen sollen in verschiedene Richtungen rotieren um eine möglichst dynamische Oberfläche des Neutronensterns zu simulieren.

7. Beleuchtung

Nun soll die Szene beleuchtet werden. Vorgesehen ist eine Punktlichtquelle, welche von der Sonne ausgeht und eine Punktstrahler, der sich auf einem der Habitate befindet und einen eingeschränkten Abstrahlwinkel hat. Das Sonnenlicht sollte eine variable Intensität haben und die Intensität des Punktstrahlers sollte von der Sonnenintensität abhängen. Der Punktstrahler soll dabei immer ins Zentrum des Systems zeigen.

Erweitern Sie die Benutzeroberfläche um Schieberegler für die Sonnenintensität und die Intensität des Punktstrahlers.

Wichtige Funktionen: `glEnable()`, `glLightfv()` mit den Parametern `GL_POSITION`, `GL_DIFFUSE`, `GL_AMBIENT`, `GL_SPOT_CUTOFF`, `GL_SPOT_EXPONENT`

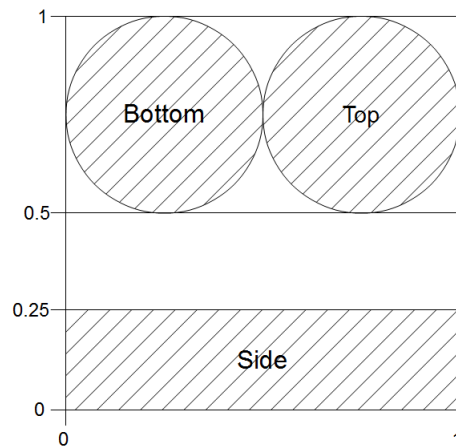


Abbildung 2: *Texturkoordinaten der Habitate*

8. Animation

Im letzten Schritt sollen die Körper animiert werden. Es gibt dabei 3 Bewegungen: Die Animation des Neutronensterns, die Bewegung der Objekte auf ihren Kreisbahnen und die Eigenrotation der Habitate.

Durch einen bereits implementierten Timer wird die `RenderWindow::timerUpdate-Methode` in regelmäßigen Abständen aufgerufen. Diese Funktion sollte daher alle zeitlichen Abläufe steuern. Die Benutzeroberfläche sollte um einen Schalter erweitert werden, mit dem dieser Timer gestoppt und wieder gestartet werden kann, für den Fall, dass man sich die Szene einmal ohne Animationen ansehen möchte.

Außerdem soll die Bahngeschwindigkeit und die Anzahl der Eigenrotationen der Habitate pro Umlauf über die Benutzeroberfläche angepasst werden können. Dabei ist zu beachten, dass sich das letzte Habitat, welches als Lichtquelle dient, nicht um sich selbst drehen soll.

Die Animationen werden dabei durch die Transformationsmatrizen umgesetzt.

Als optisches Gimmick sollen sich die Kugeln des Neutronensterns auf unterschiedlichen Achsen um sich selbst drehen.

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.