Praktikum Computergraphik

– Übungsblatt 1 –

Lehrstuhl für Computergraphik und Multimediasysteme

Julian Bader, Jonathan Klein

Überblick

In dieser Aufgabe soll ein Karussell mit mehreren Armen modelliert werden.

Die Arme sind an der Hauptsäule befestigt, die sich in der Mitte des Fundamentes befindet. An den Armen sind sich drehende Sitze befestigt, zusätzlich bewegen sich die Arme hoch und runter.

Die Szene wird von zwei Lichtquellen beleuchtet, eine simuliert das Sonnenlicht, eine andere ist eine Punktlichtquelle, die sich über einem der Sitze befindet.

Über die grafische Benutzerschnittstelle des Programms können diverse Parameter des Karussells, 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 buntes Dreieck 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 Karussell 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. Die Größe der Kugel sollte durch das Mausrad 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 Karussell wird nur aus 2 verschiedenen Objekttypen zusammengebaut - Kugeln und Zylinder. Da diese häufig und in sehr verschiedenen Formen 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 Karussell

Mithilfe der Zylinder und Kugeln wird nun das Karussell modelliert. Das Fundament besteht dabei aus einem flachen Zylinder mit großem Durchmesser, in dessen Mitte die Hauptsäule in Form eines schmaleren und hohen Zylinders steht. An Ende der Hauptsäule befindet sich eine Kugel, an der die Arme befestigt sind.

Die Anzahl der Arme ist dynamisch und soll über die Benutzeroberfläche eingestellt werden können (achten Sie dabei auf sinnvolle Minimal- und Maximalwerte). Jeder Arm besteht aus einer Stange, an dessen Ende wieder eine Kugel ist. An dieser Kugel ist mit einer weiteren Stange der Sitz befestigt, der wiederum auf einem flachen Zylinder besteht.

Um die Szene aufzubauen sollte der Matrix-Stack von OpenGL benutzt werden. Außerdem sollten die Arme des Karussells eine zufällige Farbe besitzen.

Wichtige Funktionen: `glLoadIdentity()`, `glPushMatrix()`, `glPopMatrix()`, `glRotatef()`, `glTranslatef()`, `glScalef()`, `glColor3f()`

6. Texturierung

Die benötigten Texturen für das Karussell 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 'resources.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.

Da insbesondere die Zylinder sehr unterschiedliche Ausmaße haben, können nicht alle Instanzen der Zylinderklasse die selben Texturkoordinaten benutzt. Daher sollten die Texturmatrizen benutzt werden, um die Texturkoordinaten anzupassen. Die Funktionsweise ist identisch zu den anderen Matrizen, es muss lediglich der Matrix-Mode passend gesetzt werden.

7. Beleuchtung

Nun soll die Szene beleuchtet werden. Vorgesehen ist eine direktionale Lichtquelle, welche das Sonnenlicht simuliert und eine Punktlichtquelle, die sich auf einem der Sitze befindet. Das Sonnenlicht sollte eine variable Intensität und Richtung haben und das Punktlicht nur eine beschränkte Reichweite, um den Eindruck einer lokalen Lichtquelle zu verstärken. Außerdem sollte das Punktlicht farbig sein.

Erweitern Sie die Benutzeroberfläche um Schieberegler für die Sonnenrichtung, die Sonnenintensität und Schalter um das Sonnen- und Punktlicht zu aktivieren bzw. zu deaktivieren.

Wichtige Funktionen: `glEnable()`, `glLightfv()` mit den Parametern `GL_POSITION`, `GL_DIFFUSE`, `GL_AMBIENT`, `GL_LINEAR_ATTENUATION`, `GL_CONSTANT_ATTENUATION`, `GL_QUADRATIC_ATTENUATION`

8. Animation

Im letzten Schritt soll das Karussell animiert werden. Es gibt dabei 3 unabhängige Bewegungen, die sich überlagern: Die Hauptsäule dreht sich, die Arme bewegen sich hoch und runter (durch eine Drehbewegung) und die Sitze drehen sich um ihre Aufhängung.

Durch einen bereits implementierten Timer wird die `Carousel::Move`-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.

Die Animationen werden dabei durch die Transformationsmatrizen umgesetzt. Dabei sollte beachtet werden, dass die Sitze unabhängig von der Höhe des Armes immer nach unten hängen.

Die Auf- und Abbewegung der Arme sollte auf einer Sinusfunktion basieren, da diese an den Endpunkten keinen abrupten Richtungswechsel beinhaltet sondern sanft abbremst und wieder in die Gegenrichtung beschleunigt. Außerdem sollten sich die Arme sich nicht immer auf der selben Höhe befinden, sondern einen gewissen zeitlichen Offset zu der Höhe des nächsten Armes haben. Dabei ist darauf zu achten, dass kein hässlicher Übergang zwischen dem ersten und letzten Arm entsteht.

Die Position der Punktlichtquelle sollte relativ zu ihrem Sitz sein, sie Dreht sich also sowohl mit ihrem Sitz um die Aufhängung des Sitzes Drehen, als auch mit dem Arm um die Hauptsäule des Karussells.

Als optisches Gimmick soll zusätzlich eine einfache Simulation der Zentrifugalkraft implementiert werden. Je nach Drehgeschwindigkeit des Karussells werden die Sitze weiter nach außen gedrückt und hängen somit nicht mehr senkrecht nach unten. Der resultierende Winkel kann dabei anhand folgender Skizze berechnet werden:

Erweitern Sie die Benutzeroberfläche um Schieberegler für die Hauptrotation und die Sitzrotation.

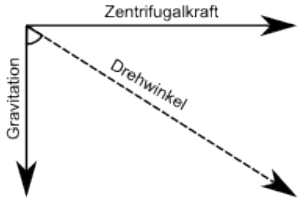

Abbildung 1: *Berechnung des Sitzwinkels*

Wichtige Funktionen: `glm::length()`, `glm::dot()`, `acos()`

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.