UNIVERSITÄT OSNABRÜCK

Dr. Thomas Wiemann

# 5. Übungsblatt zur Vorlesung „Einführung in die Programmiersprache C++“

Wintersemester 2018 / 2019

## Implementierung von Quaternionen und Vektoren

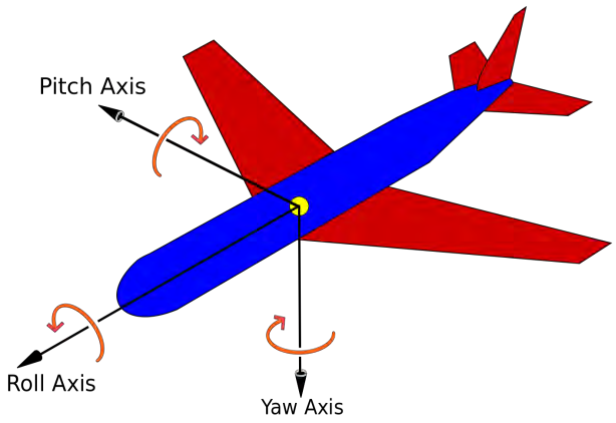
**Ziel der Übung:**

In dieser Übung lernen wir das Überladen von Operatoren in C++.

**Hintergrund:**

In der derzeitigen Version ist unser Viewer-Programm in der Lage Dreiecksnetze zu rendern. Zum Betrachten der Objekte wird eine sogenannte "virtuelle Kamera" verwendet. Damit lassen sich die Objekte aus verschiedenen Richtungen betrachten, sie bleiben aber statisch. In dieser Übung werden wir unseren Viewer derart erweitern, dass das geladene Mesh in seinem eigenen Koordinatensystem rotiert und bewegt werden kann. Bildlich gesprochen werden wir dem Fliegermodell das fliegen beibringen.

Zur Darstellung der Rotationen werden wir als mathematisches Modell Quaternionen verwenden (der mathematisch Interessierte Programmierer wird dazu bei Wikipedia fündig). Sobald der Benutzer eine Rotation anfordert, werden die Basisvektoren des Modells entsprechend transformiert. Mit Hilfe dieser neuen Basis transformieren wir alle Vertices aus dem neuen lokalen Bezugssystem in das globale System. Das lokale Koordinatensystem des Modells ist wie folgt definiert und entspricht den internationalen Konventionen für mobile Fahrzeuge:




Quelle: http://ros-robotics.blogspot.com/2015/04/getting-roll-pitch-and-yaw-from.html

Die X-Achse zeigt dabei aus Sicht des Fahrzeugs nach vorne, die Y-Achse nach links und die Z-Achse nach oben. Entsprechend bedeute ein „Roll“ um zwei Grad eine Rotation im 2 Grad um die X-Achse (entsprechend Pitch um die Y-Achse und Yaw um die Z-Achse). Im Rahmen dieser Aufgabe werden wir die `Model`-Klasse derart erweitern, dass sie entsprechende Bewegungen unterstützt. Dazu implementieren wie zunächst Hilfsklassen zur Repräsentation von Vektoren, Matrizen und Quaternionen.

### Aufgabe 1 (65 Punkte):

Erweitern Sie die Klassen `Matrix`, `Vector` und `Quaternion`. Die Kassen `Matrix` und `Vector` sollen folgende Funktionalitäten bieten:

- Arithmetische Operationen zwischen Vektoren und Matrizen: Implementieren Sie die entsprechenden Operatoren in C++ (Matrix-Matrix-Multiplikation, Matrix-Vektor-Multiplikation, Matrix- und Vektor-Skalierung (Multiplikation und Division mit floats), Skalarprodukt, sowie Addition und Subtraktion (Vektor/Vektor und Matrix/Matrix). *Hinweise*: Die Klasse `Matrix` soll hier explizit eine 4x4-Matrix sein,

deren 16 Einträge in einem eindimensionalen Array verwaltet werden. Die Einträge in diesem Array sind in C-Order („Row Major“). D.h., der Eintrag [3][2] entspricht dem zweiten Eintrag im dritten Chunk. Bildlich gibt „Drei“ dabei die Reihe und „Zwei“ die Spalte an. Diese Vorgaben entsprechen auf den OpenGL-Konventionen, so dass wir die internen Daten direkt verwenden können. Die Klasse `Vector` besteht zwar nur aus drei Komponenten, soll aber die Multiplikation mit den 4x4-Matrizen unterstützen. Sie dürfen bei der Implementierung der Multiplikation annehmen, dass die virtuelle (nicht explizit repräsentiere Komponente) immer den Wert 1 hat. Implementieren Sie die oben aufgeführten Operatoren jeweils in zuweisender und erzeugender Form, d.h., `+` und `+=` etc.

- Schreiben Sie einen Index-Operator(`operator[]` der einen Pointer auf die n-te Reihe der Matrix gibt. Warum kann man damit einen zweidimensionalen lesenden Zugriff realisieren? Was müsste man tun, um auch einen schreibenden [][]-Operator sicher zu implementieren? Warum ist der direkte Zugriff auf den Pointer aus Sicherheitsgründen bedenklich?
- Implementieren Sie einen Copy-Konstruktor und einen Zuweisungsoperator.
- Die Klasse Quaternion soll lediglich die Multiplikation mit anderen Vektoren und Quaternionen unterstützen. Die entsprechenden Berechnungsvorschriften finden Sie für unsere Koordinatensystem z.B. unter[1]

Stellen Sie sicher, dass Ihre Klassen so effizient wie möglich implementiert werden. Sorgen Sie auch dafür, dass sie bestmöglichen Schutz vor unerwünschten Seiteneffekten bieten, indem Sie Referenzen und `const`-Ausdrücke wann immer möglich verwenden. Am besten testen Sie ihre Implementation mit einem geeigneten Testprogramm.

## Aufgabe 2: Transformationen des Models (35 Punkte)

In dieser Aufgabe nutzen wir die oben implementierten Operatoren, um eine Transformation des geladenen Modells zu realisieren. Dazu finden Sie im der Klasse `Model` die entsprechenden TODO-Anweisungen. Ihre Aufgabe besteht darin, die Methoden `turn` und `move` fertig zu implementieren. Die Methode `turn` soll je nach Wert der übergebenen Aktion eine Yaw-, Pitch- oder Roll-Bewegung ausführen. Bei jeder Anwendung wird das lokale Bezugssystem des Modells bestehend aus x-, y- und z-Achse modifiziert. Erzeugen Sie dazu ein geeignetes Quaternion, dass die entsprechende Rotation repräsentiert, indem Sie den Zwei-Argumente-Konstruktor

```
Quaternion::Quaternion(const Vector& vec, float angle)
```

benutzen. Wenden Sie diese Rotation dann mittels der in Aufgabe 1 implementierten Operatoren auf die nicht zu modifizierenden Achsen an. *Beispiel*: Bei der Aktion „Roll“ soll um die X-Achse rotiert werden. Demnach wird sich diese Achse des Raumschiffkoordinatensystems nicht ändern. Die Roll-Bewegung modifiziert aber die Ausrichtung der anderen beiden Achsen des lokalen Bezugssystems, so dass diese mit Hilfe von Quaternionen rotiert werden müssen.

In der Methode `move` aktualisieren Sie die Position des Raumschiffs, indem sie es um `speed`-Einheiten in Richtung des entsprechenden Basisvektors bewegen (Accel = Bewegung in aktuelle X-Richtung, Strafe = Bewegung in Y-Richtung und Lift = Bewegung in Z-Richtung).

Damit diese Berechnungen beim Rendern des Raumschiffes berücksichtigt werden, müssen Sie noch die Methode `computeMatrix` implementieren, die aus der Lage des Basisvektoren und der aktuellen Position die Transformationsmatrix bildet und diese in `m_transformation` speichert. Diese Matrix ist wie folgt aufgebaut:

$$\begin{pmatrix} x_1 & y_1 & z_1 & p_1 \\ x_2 & y_2 & z_2 & p_2 \\ x_3 & y_3 & z_3 & p_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Dabei sind $x_n$, $y_n$, $z_n$ die Komponenten der entsprechenden Basisvektoren und $p_n$ die Komponenten der aktuellen Position (Translationsanteil). *Hinweis:* Diese Darstellung entspricht den allgemeinen Konventionen und ist daher „Column-Major“, die Vektoren sind Spaltenvektoren. Die Kodierung in C/C++ ist wie oben beschrieben „Row Major“! Berücksichtigen Sie das bei Ihrer Implementierung. Die Gleichungen für die zu implementierenden Quaternionoperationen finden sie unter diesem Link.

Wenn Sie die geforderten Funktionalitäten korrekt implementiert haben, sollte sich der Flieger mir „`wasd`“ auf der Tastatur bewegen und mit den Pfeiltasten rotieren lassen. Die Kamera wird mit der Maus bewegt.

## Abgabe:

Checken Sie Ihre Lösung des Aufgabenblattes bis Montag den 26.11.2018, 8:00 Uhr in den Master-Branch des git-Repositorys Ihrer Gruppe ein. Bringen Sie pro Gruppe einen Ausdruck des von Ihnen erstellten Codes mit.

---

`http://moddb.wikia.com/wiki/OpenGL:Tutorials:Using_Quaternions_to_represent_rotation`