Programmierübung 5: Szenenverwaltung
Computergrafik, Herbst 2016

Abgabedatum: Donnerstag 1. Dezember, 12:00

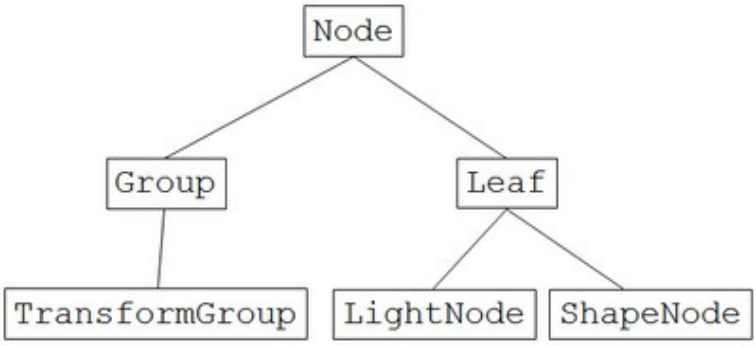
In diesem Projekt geht es darum, die Szenenverwaltung in Ihrer Rendering Engine um einen Szenengraphen zu erweitern. Die Abgabe des Projekts erfolgt wie üblich per Ilias bis am Donnerstag 1. Dezember um 12:00 Uhr, sowie direkt an die Assistenten gemäss Einschreibeliste.

1 Scene Graph

In dieser Aufgabe erstellen Sie eine weitere Implementierung der SceneManagerInterface Schnittstelle, so dass eine Graphenstruktur zur Szenenverwaltung verwendet wird. Wir nennen die neue Klasse GraphSceneManager. Orientieren Sie sich dabei an der bisherigen Implementation SimpleSceneManager, welche eine lineare Liste von 3D Objekten speichert. Implementieren Sie zuerst eine Klassenhierharchie, welche für die Knoten im Szenengraph verwendet wird.

Klassenhierarchie (2 Punkte)

Implementieren Sie ein Interface Node für Knoten im Szenengraph, und abstrakte Unterklassen Group und Leaf ähnlich wie im Bild unten gezeigt. Das Node Interface soll Methoden deklarieren, mit denen auf eine Transformationsmatrix, auf ein geometrisches Objekt, und auf die Liste der Kinder des Knotens zugegriffen werden können. Die Implementationen des Node Interface sollen null zurückgegeben, falls diese Objekte nicht existieren. Die Klasse Group soll eine Liste von Referenzen auf Kinder speichern. Sie soll Funktionalität zum Hinzufügen und Entfernen von Kindern enthalten. Sie können zum Beispiel eine LinkedList aus der java.util.Collection Package verwenden, um die Kinder zu verwalten. Implementieren Sie eine Unterklasse TransformGroup von Group, und eine Unterklasse ShapeNode von Leaf. TransformGroup soll eine Transformationsmatrix enthalten, welche auf den Teilgraphen unter dem Knoten angewendet wird. Die Klasse ShapeNode soll eine Referenz auf ein 3D Objekt enthalten. Der GraphSceneManager wird eine Referenz auf die Wurzel des Graphen enthalten.



Klassenhierarchie für den Szenengraphen

Darstellung (2 Punkte)

Der Renderer traversiert die Szene mit Hilfe eines Iterators, um dann jedes Objekt darzustellen. Dies ist zum Beispiel in GLRenderContext.display ersichtlich. Damit diese Methode auch mit Ihrem neuen GraphSceneManager funktioniert, müssen Sie also auch einen Iterator zur Verfügung stellen. Orientieren Sie sich wiederum am Vorbild des SimpleSceneManager. Beachten Sie, dass die Traversierung des Graphen mit einem Iterator eine rekursive Traversierung wie auf den Vorlesungsfolien ausschliesst! Ihr Iterator muss stattdessen die Traversierung mit einem eigenen Stack ohne rekursive Aufrufe erledigen. Dabei müssen Sie auch die Transformationsmatrizen der traversierten Knoten korrekt zusammenmultiplizieren.

Roboter (2 Punkte)

Demonstrieren Sie die Funktionalität Ihres Szenengraphen, indem Sie einen Roboter konstruieren. Der Roboter soll einen Rumpf und einen Kopf sowie Arme und Beine haben. Arme und Beine sollen aus mindestens je zwei beweglichen Teilen bestehen. Sie können Quader, Kugeln oder Zylinder als Körperteile verwenden. Es ist hilfreich, den Szenengraphen zuerst auf Papier zu skizzieren. Der Wurzelknoten wird eine Transformationsgruppe sein, welche den Rumpf darstellt. Der Knoten wird mehrere Kinder haben: eines vom Typ ShapeNode um die Geometrie des Rumpfs darzustellen, und mehrere TransformGroup Knoten für den Kopf, die Arme und die Beine. Diese Knoten werden wiederum Kinder haben.

Konstruieren Sie eine Animation, die eine Gehbewegung des Roboters implementiert. Es sollten sich mindestens die Arme oder Beine bewegen, indem sie um die Hüft- oder Schultergelenke gedreht werden. Für ein realistischeres Resultat können auch die Knie- und Ellbogengelenke animiert werden. Die Animation wird erreicht, indem die Transformationsmatrizen der entsprechenden TransformGroup Knoten verändert werden. Beachten Sie, dass sich der gesamte Teilgraph zusammen bewegen soll, wenn die Transformation einer TransformGroup an der Wurzel des Teilgraphen ändert. Lassen Sie den Roboter auf einer Grundfläche im Kreis herum marschieren.

Lichtquellen im Szenengraph (1 Punkt)

Implementieren Sie eine Knotenklasse LightNode, so dass Lichtquellen im Szenengraph gespeichert werden können. Stellen Sie einen Iterator zur Verfügung, um die Lichtquellen zu traversieren wie im SceneManagerInterface deklariert. Der Iterator funktioniert gleich wie vorher, ausser dass nur die Lichtquellen zurückgegeben werden. Sie können das instanceof Schlüsselwort verwenden, um zwischen Lichtquellen und anderen Knoten zu unterscheiden. Demonstrieren Sie diese Funktionalität, indem Sie eine Lichtquelle an einem beweglichen Teil des Roboters anbringen, z.B. an der "Hand" des Roboters.

2 Object-Level Culling mit Bounding Spheres (3 Punkte)

Fügen Sie zu Ihrer Shape Klasse eine Methode hinzu, welche eine Bounding Sphere berechnet. Dazu muss das Zentrum und der Radius berechenet werden, welche als Klassenvariablen gespeichert werden.

Erweitern Sie die Klasse ShapeNode, um View Frustum Culling mit Bounding Spheres zu ermöglichen. Falls die Bounding Sphere eines Objekts komplett ausserhalb des View Frustum liegt, muss das Objekt nicht dargestellt werden. Der Culling Algorithmus soll eine Hilfsfunktion verwenden, welche testet, ob eine Kugel komplett über oder unter einer gegebenen Ebene liegt.

Testen Sie Ihre Implementation mit einer Szene, die aus mindestens hundert Kopien des teapot oder eines anderen Objekts besteht. Sie können die Objekte zum Beispiel auf einem zweidimensionalen Gitter verteilen. Stellen Sie sicher, dass viele Kopien ausserhalb des View Frustum liegen, so dass Sie den Effekt des Object-Level View Frustum Culling testen können.

Verwenden Sie den Szenengraphen, um die Kopien mit Hilfe von Referenzen auf ein einziges Objekt zu generieren. Vermeiden Sie, das selbe Objekte mehrmals aus der Datei zu laden, oder mehrere Kopien des Objekts im Hauptspeicher zu halten. Vergleichen Sie die Bildrate mit und ohne Object-Level Culling.