Aufgabe 2: Scenegraph Basics

1 Basisimplementierung

Implementieren sie die Notwendigen Klassen, um einen Graphen zu repräsentieren. Legen sie dazu eine Klasse `Node` an, welche entweder eine Liste von Kindknoten enthält, oder eine Liste mit `Edges`, welche zwei `Nodes` verbinden.

Fügen sie ihrer Klasse ein Feld `transformation` vom Typ `ogl.vecmath.Matrix` hinzu (zur Verwendung dieses Typs sehen sie sich das zugehörige Interface-Definition und die Verwendung in der Klasse `RotatingCube` an). Es macht ebenfalls Sinn get und set Methoden für den Zugriff auf dieses Feld zu generieren.

Fügen sie weitere Methoden zum Hinzufügen und Entfernen von Kindknoten hinzu.

Tipp: Fügen sie ihrer Node-Klasse ein Feld `name` hinzu und überschreiben sie die `toString` Methode. Auf diese Weise können sie ihren Graphen auch ohne graphische Ausgabe visualisieren, was zur Fehlerfindung hilfreich ist.

2 Integration in das bestehende Projekt

Sehen sie sich die Klasse `RotatingCube` an im package `ogl.cube` des Eclipse Projekts aus der letzten Übung an.

Die Klasse besteht hauptsächlich aus den Methoden `init`, `simulate` und `display`. Des Weiteren werden Daten für einen Würfel definiert, wozu einige Hilfsmethoden benutzt werden. Identifizieren sie die für den Würfel relevanten Daten und extrahieren sie sie in eine eigene Klasse `Cube`, welche von ihrer in Teil 1 angelegten Klasse Node erbt. Die entsprechenden Code-Teile aus den Methoden `init` und `display` verschieben sie dazu in entsprechende (identisch benannte) Funktionen ihrer neuen Klasse. Den Zugriff auf die *Uniforms*

- `modelMatrixUniform`,
- `viewMatrixUniform` und
- `projectionMatrixUniform`

sowie die Variablen `vertexAttribIdx` und `colorAttribIdx` sollten sie vorerst per Getter bzw. Setter (welche in die Klasse `RotatingCube` eingefügt werden müssen) realisieren.

Testen sie ihre Implementierung und stellen sie sicher, dass das Ergebnis der initialen Implementierung in der Klasse `RotatingCube` gleicht.

Tipp: Testen sie nach Änderungen regelmäßig ob die Funktionalität der Anwendung noch besteht, da es meist schwierig ist einen Fehler nach mehreren ungetesteten Änderungen aufzuspüren.

3 Weitere Modularisierung

Nachdem die zur Implementierung des Würfels notwendigen Daten extrahiert wurden enthält die Klasse `RotatingCube` neben der grundlegenden Funktionalität zur Initialisierung von OpenGL noch Definitionen für die verwendeten Shader (insbesondere Zugriffe auf *Uniforms*).

Extrahieren sie diese Teile in eine weitere Klasse `Shader`. Sie können dazu die in Teil 2 erstellten Getter und Setter in die Shader-Klasse übernehmen.

Stellen sie abschließend eine Verbindung zwischen der Klasse `Cube` aus Teil 2 her, indem sie eine Methode `setShader` implementieren, die den für den Würfel zu verwendenden Shader festlegt. Der Shader sollte dann in der `display`-Methode des Würfels aktiviert werden (wie es zuvor in der Klasse `RotatingCube` der Fall war).

Testen sie ihre Implementierung und stellen sie sicher, dass das Ergebnis der initialen Implementierung in der Klasse `RotatingCube` gleicht.

Tipp: Kopieren sie den Shader-Code (`vsSource` und `fsSource`) in Textdateien und laden sie diese im Konstruktor ihrer Shader Klasse. Das Debugging und die Erweiterung der Shader im Java-Code ist sehr mühselig und zeitaufwändig.