Ludwig-Maximilians-Universität München
LFE Medieninformatik
Prof. Dr. Andreas Butz
Hendrik Richter, Raphael Wimmer

Übungsblatt 5: Szenegraph

Abgabe:
Dieses Übungsblatt ist als Gruppe zu lösen. Die Lösung ist bis **Dienstag,** den **8. Juni 2010, 12:00 Uhr s.t.** über UniWorx (http://www.pst.ifi.lmu.de/uniworx) abzugeben.
Es werden nur die Formate PDF und Plain-Text (UTF-8) akzeptiert. Erstellen Sie für jede Aufgabe ein Unterverzeichnis nach dem Schema <Übungsblatt>-<Aufgabe>, d.h. die Lösung der ersten Aufgabe kommt in ein Verzeichnis 5-*1/*. Packen Sie alle Dateien in eine ZIP-Datei und laden Sie diese bei UniWorx hoch. Wenn Sie Formatierungsvorgaben nicht einhalten, werden bis zu zwei Punkte abgezogen. Lösungen müssen zumindest im CIP-Pool fehlerfrei kompilieren und laufen. Bitte geben Sie nur Quellcode ab, keine kompilierten Dateien (bitte auch keine moc_xx.*-Dateien). Es können maximal 20 Punkte erreicht werden.

Aufgabe 1: Eigenschaften eines Szenegraphen (3 Punkte)

Nennen Sie drei Vorteile, die ein Szenegraph gegenüber einer naiven Implementierung rein mit OpenGL-Befehlen hat.

Geben Sie eine Datei *szenegraph.txt* oder *szenegraph.pdf* im Unterverzeichnis *5-1/* ab.

Aufgabe 2: Implementierung eines Szenegraphen (12 Punkte)

Implementieren Sie einen Szenegraphen in C++. Dieser soll möglichst flexibel sein und in der Lage sein, sich selbst rekursiv (top-down) zu zeichnen. Eine klare Struktur ist uns wichtiger als maximale Effizienz.
Implementieren Sie dazu eine Klasse **Scene**, die die ganze Szene kapselt und den Root-Node kennt. `Scene::render()` zeichnet dann den Szenegraph.
Die Einrichtung des Kontexts ist dabei nicht zwingend Aufgabe der Scene. Eine Scene enthält genau einen Root-Node vom Typ Node (s.u.). An diesem hängen alle anderen Nodes. Jeder Node kennt seinen Parent-Node und kann beliebig viele andere Nodes als Kinder haben.
Der Szenegraph wird gezeichnet, indem man rekursiv vom Root-Node durch die einzelnen Nodes geht. Speichern Sie jeweils die aktuelle Transformationsmatrix auf dem Stack bevor Sie einen Sub-Baum zeichnen und stellen Sie diese danach wieder her.
Die **Node**-Klasse ist die Oberklasse für alle Knoten (Nodes) des Szenegraphen. Diese erben von Ihr und überschreiben die Methode `applySelf()`. In der Node-Klasse ruft eine Methode `apply()` die `applySelf()`-Methoden der Kindknoten auf. Zum Beispiel so:

```
void Node::apply(){
      applySelf(); // first draw own object/apply own transformation
      for (unsigned int i = 0; i < children.size(); i++){
            glPushMatrix();
            children[i]->apply();
            glPopMatrix();
      }
}
```

Ludwig-Maximilians-Universität München
LFE Medieninformatik
Prof. Dr. Andreas Butz
Hendrik Richter, Raphael Wimmer

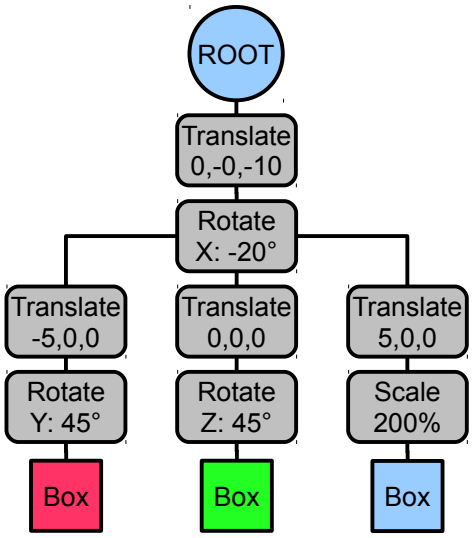
TransformNodes verwenden `applySelf()` zum Anwenden einer von ihnen gekapselten Transformationsmatrix auf die Model-View-Matrix. Von der Klasse TransformNode sollen die spezialisierten Klassen **TranslateNode**, **ScaleNode** und **RotateNode** erben. Diese sollen sinnvolle Konstruktoren haben.
GeometryNodes nutzen `applySelf()` dazu, 3D-Objekte zu zeichnen. Implementieren Sie ein Unterklasse **BoxNode**, die einen Würfel mit Kantenlänge 1 zeichnet. Diesem soll eine Farbe zugewiesen werden können.

Implementieren Sie die Node-Klasse auf Basis der folgenden Header-Datei. Kleine Änderungen sind in Ordnung.

```
#ifndef NODE_H
#define NODE_H
#include <vector>
#include <string>

class Node{
public:
	Node(std::string name = "Node");
	virtual ~Node();
	Node* getParent();
	std::vector<Node*> getChildren();
	void addChild(Node *child);
	std::string getType();
	std::string getName();
	void apply();
protected:
	std::string name;
	std::string type;
	Node *parent;
	std::vector<Node*> children;
	void setParent(Node*);
	virtual void applySelf();
private:

};
#endif // NODE_H
```



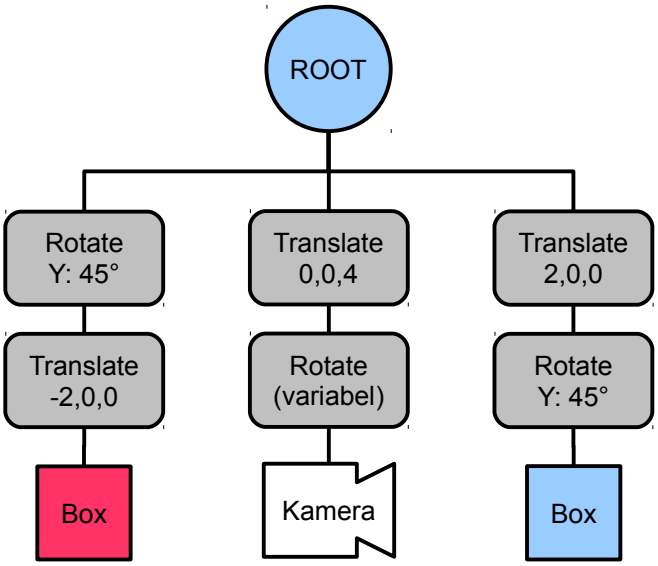
Erzeugen Sie zum Test und zur Demonstration des Szenegraphen eine QT-Anwendung, die ein OpenGL-Widget enthält. In diesem soll der obenstehende Szenegraph gezeichnet werden (die Kamera ist dabei auf 0,0,0).

Geben Sie ein Projekt *scenegraph.pro* inkl.
Quellcode und Headern in einem Unterverzeichnis *5-2/* ab.

Ludwig-Maximilians-Universität München
LFE Medieninformatik
Prof. Dr. Andreas Butz
Hendrik Richter, Raphael Wimmer

Aufgabe 3: Freie Kamera (7 Punkte)

Erweitern Sie den Szenegraph aus Aufgabe 2 dazu um einen beliebig platzierbaren **CameraNode**. Erzeugen Sie eine QT-Anwendung, die ein OpenGL-Widget enthält. Im OpenGL-Widget wird eine Szene angezeigt (siehe Abbildung), die aus einem roten und einem blauen Würfel besteht. Die Kamera befindet sich nicht im Ursprung, sondern im Punkt (0,0,4).



Die Kamera soll beliebig um ihre drei lokalen Achsen rotiert werden können. Dies kann durch drei Slider-Widgets oder durch Draggen mit der Maus implementiert werden (siehe HelloGL-Beispiel). Bei einem Rechtsklick auf das OpenGL-Widget erscheint ein Kontextmenü, in dem man einen der beiden Würfel auswählen kann (“rot oder “blau”). Die Kamera rotiert daraufhin (ohne Animation) auf den ausgewählten Würfel, so dass dessen Mittelpunkt im Mittelpunkt des Bildes ist.
Um die Kamera auf einen der Würfel auszurichten, ermitteln Sie zuerst die Positionen von Kamera und Objekt und verwenden `gluLookAt()` zum Setzen der Kameraausrichtung. Die Normale der Kameraachse ist dabei (0,1,0).
Geben Sie ein Projekt *camera.pro* inkl. Quellcode und Headern in einem Unterverzeichnis *5-3/* ab.

Viel Erfolg.