HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Übungen zur Vorlesung Computergrafik 1
- Wiederholung -

Sommersemester 2018

Prof. Dr. Ingo Ginkel

Aufgabe 1: Kennenlernen des Grund-Systems

Hinweis Aufgabe ist unverändert.

Machen Sie sich mit der Funktionsweise des gegebenen Grund-Systems vertraut. Nach dem Model-View-Controller Konzept gibt es bereits vordefinierte und implementierte Klassen im Bereich View, d.h. es existieren Methoden die Abstrakte Liste von Punkten, Polygonen etc. in OpenGl Kommandos übersetzt. Ebenso ist das OpenGl-Rendering system für den hinteren Teil der Pipeline bereits fertig. Um ein Objekt zu zeichnen benötigen sie also eigene Modelle und Kontroll-Strukturen, die die Methoden aufrufen, die dann die OpenGl-Kommandos ausführen. Sie brauchen also keine Kenntnisse über OpenGl. Es existiert bereits eine Klasse `CgSceneControl` in der Sie später ihre Objekte und Strukturen anlegen, verwalten resetten und löschen. Dies stellt den Bereich „Control“ des MVC Konzeptes dar. Die Klasse `CgQtGui` fungiert als Interface für den „View“. Beide Bereiche sind über das Observer-Pattern miteinander gekoppelt. Das bedeutet `CgQtGui` schickt alle Events (Maus-Klick, Button Press, Slider Werte geändert,...) an `CgSceneControl`. Dort ist dann die Logik vorhanden (also durch Sie zu implementieren), was als Reaktion darauf zu tun ist.

- Recherchieren Sie die Funktionsweise des Observer-Patterns und schauen Sie sich die entsprechende Pattern-Funktionalität in den Klassen `CgObserver`, `CgObservable` sowie die Anwendung des Patterns in den Klassen `CgSceneControl` und `CgQtGui` an.

Aufgabe 2: Eigene Events an den Controller weiterleiten

Hinweis Aufgabe ist unverändert.

Vorbemerkung: Die für Sie relevanten Anknüpfpunkte befinden sich alle in der Klasse `CgQtGui`. Diese Klasse realisiert die graphische Bedien- und Anzeige-Oberfläche. Hier werden die benutzten GUI-Elemente angelegt und verwaltet.

Im Sinne der späteren Änderbarkeit des Systems ist wichtig, dass keine Qt Events an ihre Controller Klasse weitergeleitet werden, sondern diese sauber abgekapselt sind und Sie eine eigene Event-Klassen-Struktur erschaffen. Es soll in der Klasse `CgQtGui` von der Gui-Logik auf die Programm-Logik abgebildet werden.

Aufgabenstellung: Implementieren Sie Gui-Elemente die später die Farbe eines Objektes ändern sollen. Fügen Sie dazu entsprechende Elemente auf das schon vorhandene TabWidget hinzu. Jeder Wert soll zwischen [0..256] bleiben, falsch-Einstellung abgefangen werden. Jedes Gui-Element sendet ein so genantes `signal` wenn es geändert wurde, das in `CgQtGui` in einem entsprechenden `slot` gefangen wird. Von dort soll dann ein selbst implementiertes Event

(z.B. CgColorChangeEvent, abgeleitet von CgBaseEvent) über das Observer Pattern an den Controller gesendet werden.

- Machen Sie sich mit dem Signal-Slot Konzept von Qt vertraut. Im Wesentlichen ist es ein Eventhandling, bei dem sich ein Listener für ganz spezielle Events anmeldet und diese bei Auftreten dann erhält.
  Infos finden Sie z.B. hier:`https://doc.qt.io/qt-5/signalsandslots.html`.
  Entsprechende Funktionalität ist für die Beispiel-Widgets (Button) schon implementiert, schauen Sie sich auch diese Beispiele an.

- Sie Können sich auch die Funktionsweise und Definition für die selbst zu erstellenden Events für die Maus- und Tastatur-Events anschauen, das ist ebenfalls schon implementiert

Geben sie in `CgSceneControl` als Reaktion auf ein Event-notify zunächst nur mit `cout` aus, welcher Event-Typ angekommen ist und welcher Farb-Wert gewünscht wird. Die eigentliche Funktionalität wird in der nächsten Aufgabe angewendet

Aufgabe 3: Erstes Objekt: Würfel

Hinweis Aufgabe ist in Teilen verändert. Farbe wird jetzt direkt in Shader übertragen.

Vorbemerkung: Als Vorlage für diese Aufgabe ist die Klasse `CgExampleTriangle` implementiert. Diese zeichnet zwei Dreiecke als Indiziertes Dreiecks-Netz. Kopieren Sie diese in eine neue Klasse und ändern Sie entsprechend ab.

Aufgabenstellung:

a) Implementieren Sie eine Klasse, die einen Einheitswürfel zentriert um den Koordinaten-System Ursprung realisiert. Der Würfel soll als polygonales Netz ausschließlich aus Dreiecken realisiert werden. Die 8 Ecken sind also die zu indizierenden Punkte für 12 Dreiecke. Leiten Sie ihre Klasse von `CgBaseTriangleMesh` ab und implementieren sie die benötigten virtuellen Methoden. Mit tatsächlicher Funktionalität müssen die Methoden für das Auslesen der Punkte, Vertex-Normalen und der Indices versehen werden (`CgExampleTriangle` dient hier als Vorlage). Instanziieren Sie ihre Klasse, so dass die Würfel-Daten erzeugt werden. Achten Sie dabei auf die Orientierung, d.h. wo Innen und Außen ist

Hinweis: Von der jeweiligen Außenseite einer Würfelseite aus gesehen müssen die Indices der Punkte gegen den Uhrzeigersinn orientiert sein.

b) Überprüfen Sie die Orientierung ihrer Dreiecke indem Sie für jedes Dreieck die Normale an den Schwerpunkt des Dreiecks zeichnen. Dazu ist eine Klasse `CgPolyline` zu implementieren die von `CgBasePolyline` ableitet. Setzen Sie den Schwerpunkt als Anfangspunkt und Schwerpunkt+Normalenrichtung als Endpunkt für die Linie. Es sind also 12 Instanzen dieser Klasse zu erzeugen, um die Normalen zu zeichnen.

c) Koppeln Sie die in Aufgabe 2 implementierte Farb-Wechsel-Funktionalität mit der Farbe der Objekte. Diese sollen beim Rendering jeweils ausgelesen und für das aktuell zu zeichnende Objekt verwendet werden. Übertragen Sie diese beim Zeichnen mit den

Funktionen `setUniformValue(...)`, so dass diese im Shader-Programm genutzt werden können und auf den entsprechend gewünschten Wert eingestellt werden. In der Klasse `CgSceneControl` kann dann mit `m_renderer->redraw()` ein entsprechendes Update angestoßen werden.