



---

## Aufgabenblatt 6

04.06.2018

### Aufgabe 13: Material, Farbe und Licht

Implementieren Sie eine Klasse `CgAppearance` um Material-Eigenschaften und andere Attribute für Objekte im Szenegraphen zu verwalten.

- a) Implementieren Sie eine Klasse `CgAppearance`, die die Farb- und Material-Eigenschaften eines Objektes verwaltet. Als Material-Parameter sollen alle Parameter verwaltet werden, die beim Phong-Beleuchtungsmodell benötigt werden. 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 (vgl. nächste Aufgabe).
- b) Implementieren Sie zusätzlich eine RGB-Farbe pro Objekt, die dann verwendet wird, wenn die Beleuchtung ausgeschaltet ist (ggf. eigenen Event dafür implementieren).
- c) Implementieren Sie entsprechende GUI-Elemente und Events, so dass Material entweder aus einer vordefinierten Liste (die Materialien aus den VL-Folien) oder frei (d.h. durch setzen der entsprechenden Werte) festgelegt werden kann. Die Material-Events sollen von Ihrem Szenegraphen verarbeitet werden und auf das jeweils gerade selektierte Objekt angewendet werden (analog für die Farbe ohne Beleuchtung).
- d) Implementieren Sie eine Klasse zur Repräsentation einer Punktlichtquelle (Instanz z.B. als Teil des Szenegraphen in Welt-Koordinaten) und übertragen Sie Position und Eigenschaften (alle die für Phong-Beleuchtung benötigt werden) wieder mit `setUniformValue(...)` an die Shader-Programme.
- e) Implementieren Sie entsprechende GUI-Elemente und Events, so dass Position und Eigenschaften der Lichtquelle (es reicht wenn es genau eine pro Szene gibt) eingestellt werden können.

### Aufgabe 14: Shader Programm zur Beleuchtungsberechnung

Implementieren Sie eine Funktionalität um Objekte zu beleuchten. Schreiben Sie dazu zwei eigene Shader-Programme in GLSL (Version 130, deswegen die ältere Version weil das in Qt und auch in VMs noch funktioniert).

Sie können entweder die Dateien `simple.vert` und `simple.frag` verändern oder eigene Text-Dateien anlegen und diese in `CgQtG1RenderWidget.cpp` Zeile 238 einlesen lassen. Wahlweise können Sie auch eigene Events implementieren, die die Shader-Source-Files umschalten können (mit `void CgBaseRenderer::setShaderSourceFiles(std::string filename_vert, std::string filename_frag)`).

- a) Machen Sie sich mit der Shader-Sprache GLSL vertraut. Viele Dinge sind analog zur glm-Library gestaltet (etwa Vektoren, Matrizen und mathematische Operationen). Informationen finden Sie z.B. unter  
<https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.1.30.pdf>.
- b) Schreiben Sie einen Vertex-Shader zur Berechnung von Gouraud-Shading. Berechnen Sie also die Beleuchtung pro Vertex im Vertex-Shader (in Kamera-Koordinaten) und geben Sie die berechnete Intensität als out-Variable an den Fragment-Shader weiter.
- c) Schreiben Sie einen Fragment-Shader zur Berechnung von Phong-Shading. Geben Sie im Vertex-Shader die benötigten Werte als out-Variable an den Fragment-Shader weiter und berechnen Sie die Beleuchtung dort pro Pixel in NDCs.
- d) Implementieren Sie eine Funktionalität bei der sich per GUI pro Objekt die verschiedenen Shading-Arten (Gouraud/Phong) umschalten lassen und analog dazu die Art der Interpolation für die Normalen (flat/smooth).

Sie können diese Einstellungen bei den GUI-Elementen für Farbe und Material aus der vorherigen Aufgabe ergänzen.

**Hinweis 1:** Als Standard ist smooth (d.h. Interpolation für jeden Pixel) in der Systematik der out-Variablen beim Vertex Shader vorgesehen. Für flat-Shading wird hier hinter out noch `flat` geschrieben (und zusätzlich auch analog bei der entsprechenden Fragment-Shader in-Variable) um zu verhindern, dass die Normalen interpoliert werden.

**Hinweis 2:** Mit dieser Funktionalität kann sowohl Flat-Shading als auch Flat-Phong-Shading realisiert werden.