OpenGL / Phong Lighting

Kurz & Knapp

- Was ist der Unterschied zwischen lokaler und globaler Beleuchtung?
- Nennen Sie 3 Effekte der globalen Beleuchtung.
- Gehören Schatten zur lokalen oder globalen Beleuchtung?
- Was ist der Vorteil von Gouraud-Shading?
- In welcher Konfiguration wäre Flat-Shading ohne nennenswerte Nachteile?
- Was ist der Unterschied zwischen Phong-Lighting und Phong-Shading?
- Geben Sie Beispiele für ambiente, diffuse und spekulare Beleuchtung.
- In welche Richtung zeigt der Licht-Vektor?

Aufgabe 1

Im Tarball `cg-2021-a01.tar.bz2` finden Sie ein minimalistisches Programm zum rendern der aus der Vorlesung bekannten Atrium-Szene mit OpenGL. Anhand dessen soll das Phong Beleuchtungsmodell implementiert werden.

Libraries.

Bevor Sie mit der Übung beginnen können müssen die nötigen Abhängigkeiten installiert werden. Aus der Paketverwaltung in Debian-Derivaten (Debian, Ubuntu, Mint) sind folgende Pakete zu installieren:

```
user@debian$ apt install build-essential libassimp-dev libglew-dev libglfw3-dev libglm-dev
```

für Void Linux:

```
user@void$ xbps-install gcc make libassimp-devel glew-devel glfw-devel glm
```

Zusätzlich basiert der Code in `cg-2021-a01.tar.bz2` auf dem BigDuckGL Framework und verwendet ImGui. Um diese zu installieren gehen Sie bitte wie folgt vor:

Laden Sie sich aus GRIPS `imgui-2021.tar.bz2` herunter und entpacken Sie es, z.B. via

```
tar xf imgui.tar.bz2
```

Sie können die Bibliothek mit

```
./configure
```

passend für Ihr System konfigurieren, mit

```
make
```

übersetzen und mittels

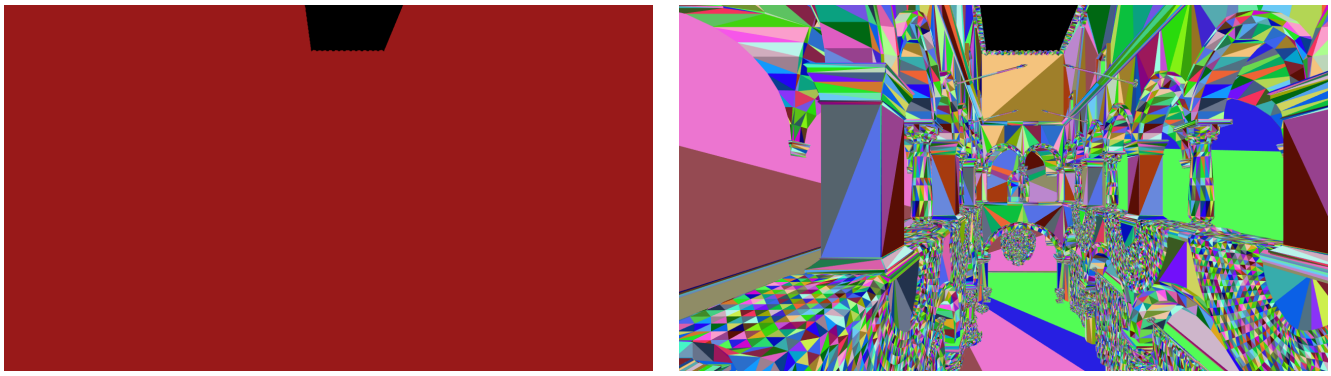
```
sudo make install
```

nach `/usr/local` installieren.

Laden Sie sich als nächstes `bigduck-2021.tar.bz2` aus GRIPS herunter und entpacken, konfigurieren, kompilieren und installieren Sie es analog zu ImGui.

Nun sollten Sie ebenfalls `cg-2021-a01.tar.bz2` auspacken und übersetzen können. Da wir an diesem Programm direkt entwickeln ist eine Installation natürlich nicht nötig. Weiterhin benötigen Sie den Tarball `models.tar.bz2` von Grips, den Sie am einfachsten direkt in `render-data` auspacken, so dass das Verzeichnis `render-data/models` entsteht. Sie können dann das Programm aus dem `cg-2021-a01`-Verzeichnis wie folgt starten:

```
user$ ./src/cg
```



Ansicht aus der Vorlage (links) und Ergebnis der ersten Teilaufgabe (rechts).

Überblick.

Das ausgeteilte Programm besteht nur aus einer C++ Datei, vieles aus der letzten Übung wird nun innerhalb von BigDuckGL erledigt. Das Programm startet mit der Erzeugung des OpenGL Kontextes. As nächstes wird die initiale Kameraposition so gewählt, dass direkt ein verwendbarer Teil der Szene angezeigt wird. Sie können sich mit `WASD` bewegen, sowie mit `RF` nach oben bzw. nach unten schweben, mit `QE` rollen, die Geschwindigkeit der Bewegung kann über das Mausrad gesteuert werden. Mit `F1` erscheint eine Leiste die es Ihnen erlaubt, z.B., einen Screenshot zu speichern.

Nach den Kameraeinstellungen werden verschiedene Shader für die Teilaufgaben geladen. Die einzelne Bedeutung wird in den Teilaufgaben beschrieben.

Im Main-Loop (`while (Context::running()...)`) werden periodisch die Shader neu geladen (d.h. Sie können live editieren, eventuelle Fehlermeldungen erscheinen im Terminal) und die UI gezeichnet. Dabei werden auch die Benutzerdefinierten Werte für Lichtpositionen etc aktualisiert und insbesondere auch die aktuelle Teilaufgabe.

Im eigentlichen Rendering-Abschnitt (nach `glClear(...)` und vor `swap_buffers()`) wird die Szene je nach Teilaufgabe unterschiedlich gerendert. Sie müssen nur in dem Bereich, sowie in den entsprechenden Shader Dateien Code anpassen/erweitern.

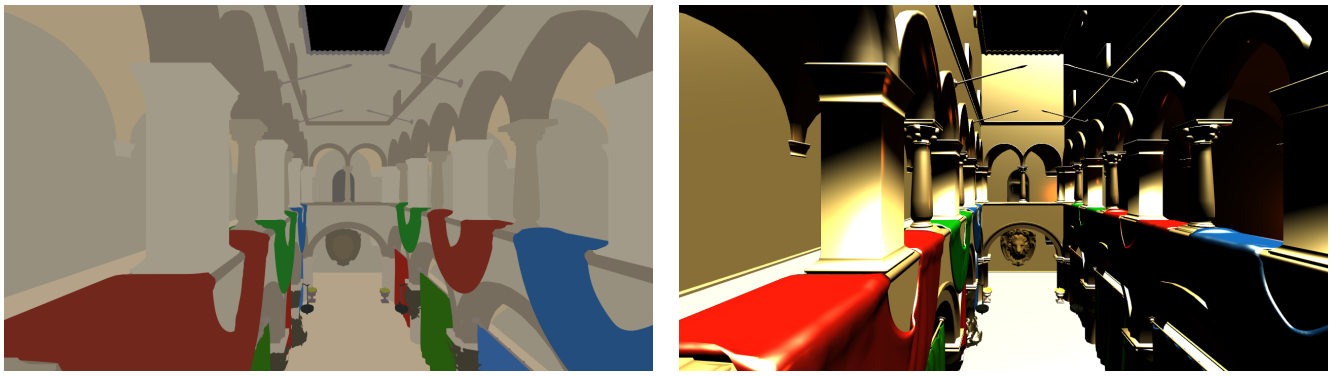
Aufgabe 1.1

Im letzten Blatt haben wir die einzelnen Dreiecke mit individuellen Farben versehen in dem ein zusätzliches Vertex-Attribut verwendet wurde. Dass diese Methode nicht besonders allgemein ist war schon Gegenstand des letzten Blattes.

In dieser Aufgabe wird ein Uniform-Array verwendet. Dazu werden vor dem Main-Loop 100 Zufallsfarben erzeugt und in einem Array abgelegt. Dieses Array kann nun mittels einer einzelnen Uniform-Variablen an den Shader übergeben werden. Wie Sie im Shader `default.frag` sehen ist die Uniform-Variable als Array spezifiziert. Übergeben Sie die Zufallsfarben an den Shader und greifen Sie mit `gl_PrimitiveID` auf das Array zu. Beachten Sie, dass Sie keinesfalls über die Arraygrenzen hinaus zugreifen dürfen!

Da dieser Fall in BigDuckGL nicht abgedeckt ist können Sie die Uniform-Variable nicht setzen wie die anderen, bereits vorhandenen. Holen Sie sich die Uniform-Location direkt mittels OpenGL (als Shader-Parameter können Sie `shader_plain_color->id` verwenden) und übergeben Sie das Array mit einem einzelnen Aufruf von `glUniform4fv`.

Hinweis. Die OpenGL Manpages (auch online verfügbar, siehe Vorlesungs-Folien) sind sehr ausführlich und genau, wenn Sie die genannten OpenGL Befehle oder Shader-Variablen recherchieren möchten sind sie ein guter Startpunkt, genauso wie das Khronos OpenGL Wiki.



Ergebnis der zweiten (links) und dritten Teilaufgabe (rechts).

Aufgabe 1.2

Das BigDuckGL Framework lädt die Szene schon mit allen Materialien. Lesen Sie sich den Code der Funktionen `Drawelement::bind`, `Drawelement::draw` und `Drawelement::unbind` aus BigDuckGL durch, schauen Sie auch in die darin aufgerufenen Framework Funktionen, insb. bzgl. der Materialeigenschaften. Die umgesetzte Methode entspricht recht gut dem in der Vorlesung besprochenen Modell des Bindens und Lösens von OpenGL Ressourcen.

Im Shader nicht verwendete Uniform-Variablen gelten nicht als Fehler, was Sie schon daran erkennen, dass das Framework einige Uniforms definiert die im Shader aus dem `MaterialColor`-Zweig nicht vorkommen (siehe `material.frag`). Ergänzen Sie den Shader so, dass auf allen Oberflächen die diffuse Materialfarbe angezeigt wird. Ändern Sie dazu nur den Shader!

Aufgabe 1.3

Über die UI können zwei Lichtquellen eingestellt werden: ein direktionales Licht (es wird die Lichtrichtung aus Sicht der Lichtquelle eingestellt, sowie die Lichtfarbe) und ein Punktlicht (dessen Position in der Szene und Lichtfarbe eingestellt werden kann). Für beide Lichter kann zusätzlich zur Farbe noch ein Skalierungsfaktor angegeben werden.

Die entsprechenden Uniform-Variablen werden im `PhongLighting`-Zweig bereits an den Shader übergeben. Implementieren Sie damit die Beleuchtung in `shading.frag`. Den Distance-Falloff des Punktlichts und die spekularen Exponenten können Sie dabei so wählen, wie Sie möchten. Beachten Sie die im entsprechenden Vertex-Shader emittierten `out`-Variablen (vollziehen Sie den Code des Shaders nach!).

Der Shadercode sieht initial vor, die Berechnung im Worldspace auszuführen. Probieren Sie es ebenfalls im Eyespace, was ist hier die Richtung zur Kamera?

Die Materialparameter in der Szene sind nicht optimal gesetzt; Sie können diese gerne anpassen. In der Datei `render-data/models/sponza/sponza.fixed.cols.mtl` sind die Materialfarben mit `Kd` und `Ks` spezifiziert, Sie können hier gerne Änderungen vornehmen (müssen aber nicht).

Aufgabe 1.4

Vorschlag wenn Sie etwas mehr sehen wollen: Fügen Sie mehrere Punktlichter hinzu, übergeben Sie die Punktlicht-Daten als Uniform-Arrays wie bei den Zufallsfarben und bewegen Sie die Lichter automatisch in der Szene. Die Lichter könnten sich auf interessanten Bahnen bewegen, möglicherweise in nicht-uniformer Geschwindigkeit, pulsieren (Intensitätsskalierung), die Farbe ändern, etc. Besonders abgefahrene Implementierungen dürfen gerne eingereicht werden :)

Happy Hacking :)