TH KÖLN
Prof. Dr. Florian Niebling
Advanced Media Institute

Technology
Arts Sciences
TH Köln

19. Juni 2024
Abgabe: spätestens am 12. Juli 2024

Computergrafik und Animation
Blatt 4 - Lights

In dem letzten Aufgabenblatt dieses Semesters werden Sie sich mit der Beleuchtung einer Szene anhand des Phong-, bzw. Blinn-Phong-Modells beschäftigen. Dies wird in der Kategorie *Materials and Light* des THe-Learning-Systems thematisiert.
Testen Sie Ihre Implementierung nach jedem Schritt, die zusammengesetzten Teilergebnisse sollen letztlich ein Renderergebnis ähnlich der Abbildung 4 ergeben. Es bietet sich an, verschiede Terme der Beleuchtungsberechnung einzeln als Farbe auszugeben um diese visuell zu prüfen.

4.1 Phong-Modell

$$L_O = \underbrace{M_e}_{A} + \underbrace{M_d \cdot L_a}_{B} + \sum_{i=1}^{n} (\underbrace{M_d \cdot \max(\cos\alpha, 0)}_{C} + \underbrace{M_S \cdot \max(\cos\beta, 0)^k}_{D}) \cdot L_i \tag{1}$$

4.1.1 Phong-Modell Terme

Die obengenannte Gleichung stellt das erweiterte Phong-Modell dar. Bitte machen Sie sich mit der Theorie vertraut und spezifizieren Sie die Buchstaben A - D bzgl. ihres Aufgabenbereiches. In Abbildung 1 sind die verwendeten Winkel visualisiert.

A -

B -

C -

D -

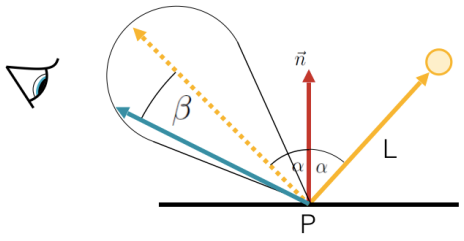

Abbildung 1: Schematische Darstellung des Phong-Modells

4.1.2 Cosinus-Terme

Überlegen Sie warum das $max(cos, 0)$ bei den Cosinus-Termen notwendig ist.

4.2 Integration einer Punktlichtquelle

Mit dem Aufgabenzettel haben Sie wieder zwei Interfaces erhalten, welche für die Beleuchtung genutzt werden sollen. Eines ist für die Implementierung einer Punktlichtquelle und eines für ein Spotlight.
Denken Sie daran, dass im Folgenden alle Berechnungen im View- bzw. Cameraspace durchgeführt werden sollen. Daher müssen auch alle benötigten Vektoren in dieses Koordinatensystem transformiert werden.

4.2.1 `PointLight`

Erstellen Sie bitte die Klasse `cga.exercise.components.light.PointLight`, welche von der Klasse `Transformable` erbt und das Interface `cga.exercise.components.light.IPointLight` nutzt. Der Konstruktor nimmt als Parameter die gewünschte (initiale) Lichtposition im Weltkoordinatensystem sowie die Lichtstärke / Farbe als RGB-Vektor.

Die Lichtfarbe soll als Eigenschaft gespeichert werden und das `PointLight` soll an der gegebenen Position im Weltkoordinatensystem platziert werden. Nutzen Sie hierzu die Funktionaliät aus der Basisklasse `Transformable` im init-Block von `PointLight`.

AuSSerdem muss die Methode `bind(...)` aus `IPointLight` implementiert werden. In dieser Methode sollen nach gewohnter Art die Parameter der Lichtquelle (Farbe und Position) an die Shader übertragen werden. Informationen zur Weltposition und -orientierung können über die Basisklasse `Transformable` abgefragt werden (sofern sie zuvor im init-Block gesetzt wurden!).

4.2.2 `Scene`

Im `Init`-Block der `Scene` muss das PointLight angelegt und mit den entsprechenden Parametern versehen werden. Bedenken Sie, dass ein Vektor $\begin{pmatrix}0 & 0 & 0\end{pmatrix}$ für die Lichtfarbe eher unpassend ist. Zudem setzen Sie die Lichtquelle als Child des Motorrads, sodass es sich mit dem Motorrad mitbewegt. Die Lichtquelle befindet sich allerdings immer noch auf gleicher Höhe mit der Ebene. Heben Sie sie etwas an, damit die Lichtquelle den Boden anstrahlen kann.
AbschlieSSend muss das Punktlicht mithilfe seiner `bind()` Methode in der `render()`-Methode der `Scene` gebunden werden. Die Reihenfolge der Operationen ist hier kritisch: Überlegen Sie sich, welche Informationen in welcher Reigenfolge von den Shadern und den verschiedenen Draw Calls gebraucht werden!

4.2.3 Shader

Für die Implementierung des Phong-Beleuchtungsmodells müssen die Shader entsprechend angepasst werden. Dort findet die eigentliche Beleuchtungsberechnung statt. Eine gute Informationsquelle ist hierfür `https://learnopengl.com/Lighting/Basic-Lighting`.

VertexShader(VS)
Im VS nehmen Sie die Position der Punktlichtquelle entgegen. Berechnen Sie die Vektoren `toCamera` und den `toLight`-Vektor für die Punktlichtquelle im Viewspace und übergeben Sie diese an den Fragment-Shader.

Hinweis: *Diese beiden Vektoren dürfen erst im Fragment Shader normalisiert werden. Warum?*
Hinweis-Hinweis: *Woher kommen die Werte in der "Mitte" eines Dreiecks, abseits der Vertices?*

FragmentShader(FS)
Führen Sie die Berechnung der Phong-Komponenten im FS durch. Hierzu verwenden Sie die Daten die vom Vertexshader übergeben werden sowie Material-Uniforms usw. Alle Richtungsvektoren müssen vor der Benutzung normalisiert werden; und nochmal: Alle Vektoren müssen sich im gleichen Koordinatensystem befinden!

Tipp: Damit die Implementierung nicht allzu chaotisch wird, bietet es sich an die verschiedenen Terme des Beleuchtungsmodells (Gleichung 1) in separate, wiederverwendbare Funktionen auszulagern.
Eine Trennung von Oberflächentermen (C, D) und einfallenden, potenziell attenuierten (Sektion 4.4.1) Lichttermen (L_i) ist semantisch besonders sinnvoll.
Sind die Einzelterme implementiert, kann man die Summe in Gleichung 1 einfach als Schleife über die Lichtquellen, oder als diskrete Schritte herunterschreiben.

4.3 Scheinwerfer

Ein Motorrad besitzt einen Frontscheinwerfer, welcher in diesem Projekt mittels eines Spotlights simuliert werden soll.

4.3.1 `SpotLight`

Hierfür erstellen Sie eine neue Klasse `SpotLight`, welche von der Klasse `PointLight` erbt und das Interface `ISpotLight` implementiert.
Charakteristisch für ein Spotlight ist, dass es gerichtet ist und somit in der Ausbreitung begrenzt. Es entstehen zwei Kegel, welche durch die abgebildeten Winkel definiert werden. Der Winkel ϕ definiert einen inneren Kegel, in dem die Fragmente voll beleuchtet werden. Für Fragmente auSSerhalb des inneren, aber innerhalb des äuSSeren Kegels schwächt sich die Beleuchtung linear ab. Dieser äuSSere Kegel wird durch den Winkel γ spezifiziert. Der abgebildete Winkel θ definiert den Winkel zwischen der Lichtrichtung der Lichtquelle und des Vektors zwischen dem betrachteten Punkt und der Lichtquellenposition. Nähere Informationen dazu finden Sie unter `https://learnopengl.com/Lighting/Light-casters`.

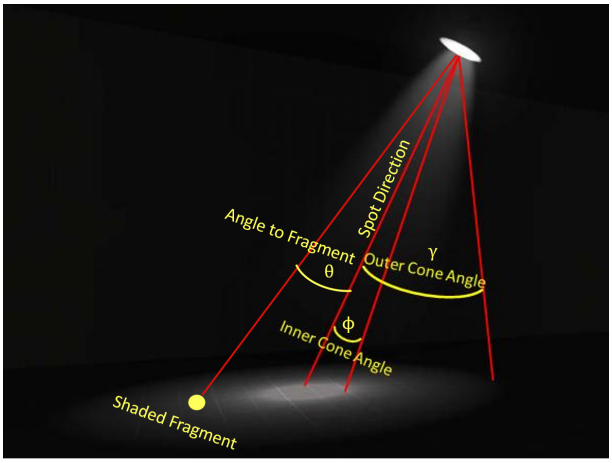

Abbildung 2: Aufbau eines Spotlights

Der Konstruktor des Spotlights beinhaltet neben den vom PointLight-Konstruktor benötigten Parametern Lichtfarbe und Position zusätzlich die Winkeldefinitionen zum inneren und äuSSeren Kegel. Diese Winkel sind private Eigenschaften des `SpotLights`. Das Interface definiert die `bind(...)`-Methode, welche von Ihnen zu implementieren ist. Die Klasse muss die Lichtfarbe, Position, die Kegeldefinition und den Richtungsvektor an die Shader übermitteln. Hierbei können Sie die Vererbung zu Hilfe nehmen. Die Winkel sollten bereits als Radiant-Werte vorliegen, um sich die Umrechnung im Fragment-Shader zu ersparen.

4.3.2 `Scene`

Von der Instanzierung bis zum RenderCall gehen Sie wie in Aufgabe 4.2.2 vor. Das Spotlight sollte sich vorne am Motorrad befinden und leicht nach unten geneigt sein, damit der Boden durch den Scheinwerfer beleuchtet wird.

4.3.3 Shader

VertexShader(VS)
Im VS nehmen Sie die Position des SpotLights entgegen und berechnen einen weiteren `toLight`-Vektor für das SpotLight.

FragmentShader(FS)
Im Fragment-Shader muss die jeweilige Lichtintensität I berechnet werden, mit der die Lichtquelle auf das Fragment strahlt. Für Fragmente im inneren Kegel ist die Helligkeit noch maximal, die Intensität ist 1. Fragmente auSSerhalb des äuSSeren Kegels werden gar nicht angestrahlt, die Intensität ist 0.
Für die restlichen Fragmente lässt sich I nach folgender Formel berechnen:

$$I = \frac{cos(\theta) - cos(\gamma)}{cos(\phi) - cos(\gamma)}$$

Achten Sie darauf, dass I zwischen 0 und 1 liegen muss. Eine nützliche GLSL-Funktion in diesem Zusammenhang ist `clamp(...)`[1].
Die ermittelte Lichtintensität kann dann mit der Lichtfarbe der Lichtquelle multipliziert werden. Übergibt man als `toLight`-Vektoren im Vertex Shader den nicht normalisierten Vektor zwischen Vertex und Lichtquelle, kann man d im Fragment Shader einfach über den Betrag des Vektors erhalten.

[1] `https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/clamp.xhtml`

4.4 Attenuation und Gammakorrektur

4.4.1 Attenuation

Bei Punktlichtquellen gilt das *Inverse Square Law*, d.h. das Licht schwächt sich mit $\frac{1}{d^2}$ quadratisch ab, wobei d die euklidische Distanz zwischen Lichtquelle und Fragment ist. Um dieses Phänomen zu simulieren, bauen Sie in die Lichtberechnung eine Lichtabschwächung (engl. Light attenuation) ein. Übergibt man im Vertex Shader den nicht normalisierten Vektor zwischen Vertex und Lichtquelle (`toLight`), erhält man im Fragment Shader die Distanz d einfach über die Länge des Vektors.

4.4.2 Gammakorrektur

Wie Sie sicherlich festgestellt haben, ist die resultierende Beleuchtung aus letzter Aufgabe etwas enttäuschend. Das liegt daran, dass wir bisher im sRGB- oder Gamma-Farbraum gearbeitet haben.

Die menschliche Farbwahrnehmung kann dunklere Farbtöne besser unterscheiden als helle. Aufgrund dessen wurde der sRGB- oder auch Gamma-Farbraum entwickelt.
Dazu wird die rote Kurve in Abbildung 3 auf die linearen Werte angewendet. Als Ergebnis werden kleine Werte über einen gröSSeren Ausgabebereich gestreckt und hellere Werte gestaucht. Das führt zu einer höheren Farbauflösung in dunkleren Bereichen.
Die Stärke der Stauchung wird mit dem Parameter γ eingestellt und ist traditionell 2.2.

Um weiterhin die korrekte Ausgabe aus dem Monitor zu erhalten, muss der Monitor die Stauchung rückgängig mache, indem er die inverse Gammakurve (blau) anwendet.
Die schlechten Ergebnisse der Beleuchtung bisher liegen an zwei Dingen:

- Die Texturen für diffuse, spekulare und emissive Farbe kommen normalerweise als sRGB und ist daher verzerrt $\Longrightarrow$ unsere Berechnungen sind falsch, weil sie von linearen Werten ausgehen!
- Wir geben (zwar falsche, aber) lineare Werte aus, die der Monitor nach unten verzerrt, weil er von sRGB Werten ausgeht $\Longrightarrow$ das Ergebnis wird zu dunkel!

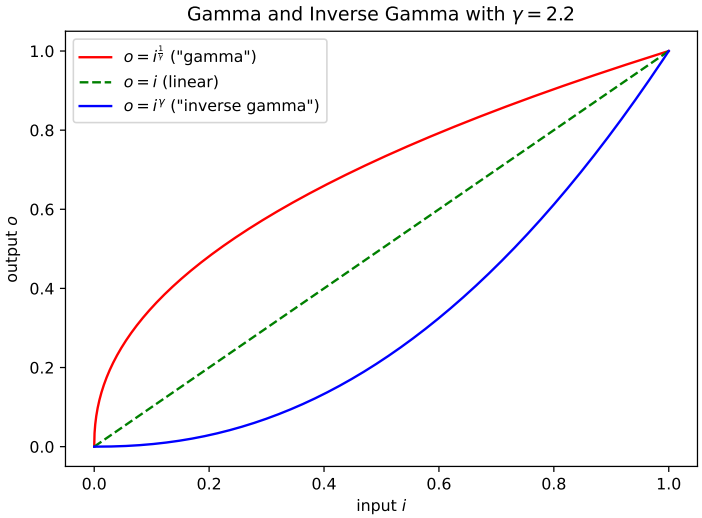

Abbildung 3: Gammakurven

Die Lösung ist sehr simpel:
Zunächst müssen wir alle Farbwerte aus den Eingabe-sRGB-Texturen (diffuse, emit, specular) $\mathbf{C}_{\text{sRGB}}$ in lineare Werte $\mathbf{C}_{\text{linear}}$ umwandeln, bevor wir mit ihnen rechnen (blaue Kurve, “inverse gamma”):

$$\mathbf{C}_{\text{linear}} = \mathbf{C}_{\text{sRGB}}^{\gamma} \tag{2}$$

Nachdem wir mit der Beleuchtungsberechnung fertig sind, müssen wir die finalen Farbwerte $\mathbf{C}_{\text{fragcol}}$ in den sRGB oder Gamma-Farbraum umwandeln, damit zusammen mit der Verzerrung des Monitors das Ergebnis wieder korrekt ist (rote Kurve, “gamma”):

$$\mathbf{C}_{\text{output}} = \mathbf{C}_{\text{fragcol}}^{\frac{1}{\gamma}} \tag{3}$$

Schreiben sie für Gleichung 2 (“inverse gamma”) und 3 (“gamma”) jeweils eine GLSL Funktion, und verwenden Sie diese wie hier beschrieben. Den Gammawert γ setzen Sie initial auf 2.2. Diesen Wert können sie aber später auch ändern, um das Ergebnis nach Ihren Wünschen anzupassen.

gamma und invgamma Funktionen

```
vec3 gamma(vec3 C_linear);      // returns value in gamma / sRGB space
vec3 invgamma(vec3 C_gamma);    // returns value in linear space
```

Hinweis: *Werden im Projekt später andere Texturarten verwendet, muss man immer schauen ob man diese als sRGB oder linear zu interpretieren hat. Texturen wie Normalmaps oder Heightmaps kommen meistens schon in linearer Form, da diese keine wirklichen Farbwerte darstellen. Das ist eine gute Daumenregel: Drückt eine Textur einen Farbwert aus muss man den Gammakorrektur-Schritt anwenden, hat man anderweitige Daten, sind diese meistens schon linear.*

4.5 Farb-Variationen

Um das Spiel ein wenig schöner und bunter zu gestalten, geben Sie beim RenderCall noch eine Farbinformation an den Shader, welche die Emission der Objekte beeinflussen soll. Für den Boden können Sie einen statischen Farbwert definieren und mit der emissiven Textur multiplizieren. Wählt man *grün (0, 1, 0)* als Farbe, färbt sich der Boden wie in Abbildung 4.
AnschlieSSend soll das Motorrad abhängig von der Zeit (`t`) ebenfalls die Farbüberlagerung erhalten, dabei aber dieselbe uniform-Variable des Bodens nutzen. Dabei lassen sich verschiedene Sinus-Werte der Zeit als RGB-Komponenten verwenden. Die Lichtfarbe des Punktlichtes soll die gleiche Farbe erhalten.
Haben Sie alle Aufgaben korrekt implementiert, ergibt sich das folgende Bild.

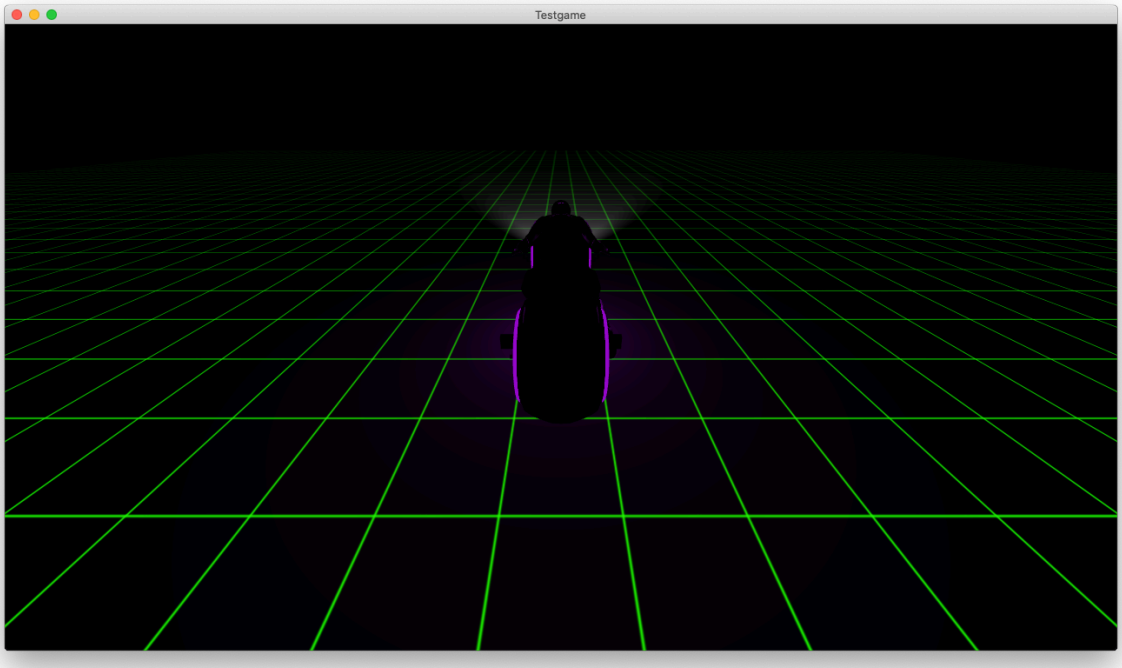

Abbildung 4: Finales Renderingergebnis

4.6 Kamera-Bewegung per Maus-Steuerung

Die Kamera soll nach dieser Aufgabe mit Hilfe der Maus um das Motorrad rotiert werden können, ohne es aus dem Fokus zu verlieren. Verwirklichen Sie diese Kamerabewegung in der Methode `Scene.onMouseMove(...)`.
Die Bewegung der Maus in x-Richtung (Differenz zwischen alter und neuer Position) stellt den Rotationswinkel um die y-Achse dar. Dieser sollte allerdings um den Faktor 0.002 skaliert werden, da die Cursor-Positionen im ScreenSpace gegeben sind.

4.7 Weitere Lichtquellen

Verteilen Sie einige weitere, verschiedenfarbige Punktlichtquellen in der Szene, beispielsweise in den Ecken der Ground Plane.

4.8 Erweiterung des Beleuchtungsmodells

Mit einer kleinen Änderung können wir den Realismus des einfachen Phong Beleuchtungsmodells deutlich verbessern. Implementieren Sie hierzu den Ansatz basierend auf den Half-Vectors (Lektion "Specular Reflection - Halfvector" 04:10). Dieses Modell wird auch "Blinn-Phong"-Modell genannt. Weitere Informationen kann man unter `https://learnopengl.com/Advanced-Lighting/Advanced-Lighting` finden.