UNIVERSITÄT BASEL

Prof. Dr. Thomas Vetter
Departement Mathematik und Informatik

Spiegelgasse 1
CH – 4051 Basel

Patrick Kahr (patrick.kahr@unibas.ch)
Clemens Büchner (clemens.büchner@unibas.ch)

Computer Grafik 2019 - Übungsblatt 4

Ausgabe in Woche 8 (11.4.2019).

Vorführung der laufenden Programme im Tutorium Woche 11 (Abgabe 02.05.2019).

Maxiaml zu erreichende Punktzahl: 15

Wir können nun Objekte auf die Bildebene projizieren und zeichnen. Bis jetzt haben wir die Farbe eines Pixels aus den Farben der Eckpunkte interpoliert. In diesem Übungsblatt implementieren Sie verschiedene Beleuchtungsmodelle und untersuchen deren Unterschiede.

Aufgabe 1 - Perfekter Lambert'scher Strahler (3 Punkte)

Wie Sie aus der Vorlesung wissen, besteht von einem Objekt abgestrahltes Licht hauptsächlich aus diffus und spekulär/spiegelnd reflektiertem Licht.

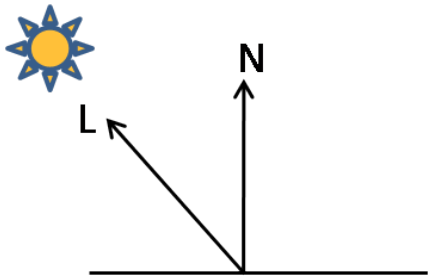
Für diese Aufgabe nehmen wir an, dass der Teapot ein perfekter Lambert'scher Strahler ist. Das heisst: Punkte auf der Oberfläche absorbieren Licht und geben dieses in alle Richtungen zu gleichen Teilen ab. Der Anteil des abgestrahlten Lichtes wird durch den Reflektions-Koeffizienten (Albedo) bestimmt. Die Menge des absorbierten und danach abgestrahlten Lichtes hängt von dem Winkel zwischen dem einfallenden Licht und der Oberflächennormalen ab. In dieser Aufgabe nehmen wir an, dass der Teapot das Licht nicht spekulär reflektiert. In Formeln sieht dies so aus:

$$I_{Lambert} = I_{Light} \cdot \langle \hat{\vec{L}}, \hat{\vec{N}} \rangle \cdot albedo$$

Dabei sind:

- I_{Light}: `RGBA lightSource.color`
- *albedo*: `double MATERIAL_ALBEDO`
- $\langle ., . \rangle$ ist ein Skalarprodukt
- $\vec{N}, \vec{L}$: siehe Abbildung
- $\hat{\vec{V}}$: ein normalisierter Vektor

UNI BASEL



Wir nehmen an, dass die Lichtquelle eine Punkt-Lichtquelle ist. $\vec{L}$ muss also aus den Positionen der Lichtquelle und des betrachteten Punktes berechnet werden.

Implementieren Sie die Methode `colorizePixel` der Klasse `LambertMeshRenderer`, welche die Pixel als perfekte Lambert'sche Strahler einfärbt.

Benötigte Dateien: renderer.LambertMeshRenderer.java
Mögliche Tests: Lambert Renderer
Hinweis Nur Flächen, welche zur Lichtquelle hin zeigen strahlen auch Licht dieser Quelle aus.

Aufgabe 2 - Phong'sches Reflexionsmodell (3 Punkte ⋆)

Wenn Sie bei dem LambertRenderer-Test den Teapot von hinten betrachten, stellen Sie fest, dass er im schwarzen Hintergrund verschwindet. In der realen Welt sind Oberflächen, welche nicht direkt von einer Lichtquelle beleuchtet werden meistens nie ganz schwarz. Dies, weil sie von Licht beleuchtet werden, welches von anderen Oberflächen reflektiert wurde. Das Phong'sche Beleuchtungsmodell approximiert dieses Phänomen, indem alle Oberflächen von einer *ambienten* Lichtquelle gleich stark beleuchtet werden. Das Phong-Modell simuliert auch spekulär reflektiertes Licht, indem es den Winkelunterschied zwischen dem reflektierten Strahl und dem Auge mit einbezieht.

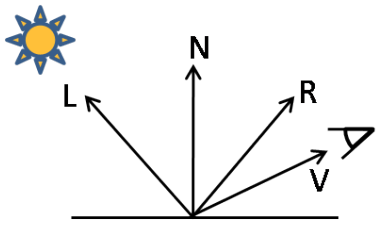
Bei uns bestehen Phong'sche Lichtquellen aus deren Position, der Farbe und einer "ambienten Lichtquelle".

In Formeln sieht dies folgendermassen aus:

$$I_{Phong} = r_A \cdot I_A + r_D \cdot I_C \cdot \langle \hat{\vec{L}}, \hat{\vec{N}} \rangle + r_S \cdot I_C \cdot \langle \hat{\vec{R}}, \hat{\vec{V}} \rangle^m$$

- I_A: `RGBA lightSource.ambient`
- I_C: `RGBA lightSource.color`
- r_A: `RGBA material.ambient`
- r_D: `RGBA material.diffuseReflectance`
- r_S: `RGBA material.specularReflectance`

- m: `double material.shininess`
- $\langle ., . \rangle$ ist ein Skalarprodukt
- $\hat{\vec{V}}$: ein normalisierter Vektor
- $\vec{N}, \vec{L}, \vec{R}, \vec{V}$: siehe Abbildung



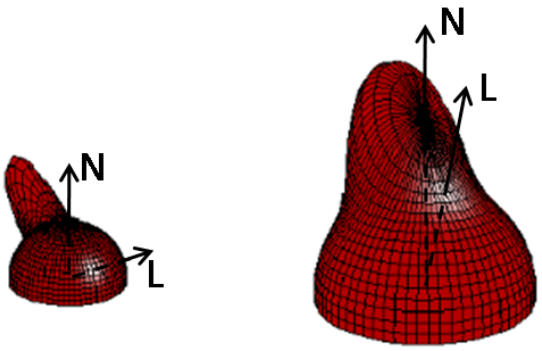
Implementieren Sie die Methode `colorize` der Klasse `PhongMeshRenderer`, welche Meshes nach dem beschriebenen Phong-Model einfärbt.

Hinweis: Welche Werte der Skalarprodukte sind sinnvoll und welche sollten besser nicht verwendet werden?

Benötigte Dateien: renderer.PhongMeshRenderer.java
Mögliche Tests: Phong Reflectance Model

Aufgabe 3 - Bidirectional Reflectance Distribution Function (BRDF) und Lambert'sche Strahler (3 Punkte $\star$)

Wir distanzieren uns etwas vom Phong'schen Reflexionsmodell und betrachten Oberflächen von Materialien etwas abstrakter. Anstelle verschiedener verhalten einer Oberfläche für (ambiente), diffuse und spekuläre Reflektion, versuchen wir das Verhalten der Obefläche mit einer Funktion zu beschreiben. Diese Funktion heisst Bidirectioanl Reflectance Distribution Function (BRDF). Die BRDF ist eine Eigenschaft der Oberfläche und nimmt als Parameter die Richtung zur einfallenden Lichtquelle, die Normalenrichtung und die Richtung zur Kamera entgegen. Das Resultat der BRDF ist die radiance/Abstrahlung unter den gegebenen Parametern.



BRDF einer Oberfläche bei fixierter Normalen- und Lichtrichtung. Der Funktionswert represäntiert die Abstrahlung in verschiedene (Blick-) Richtungen.

a) BRDF des Lambert'schen Strahlers (1 Punkt)
Implementiern Sie die Funktion `getRadiance()` der Klasse `LabertBrdf`. Der Rückgabewert dieser Funktion wird in der bereits implementierten Methode der Klasse `Brdf` mit dem $\langle \hat{\vec{L}}, \hat{\vec{N}} \rangle$ produkt gewichtet und elementweise mit der Farbe der Lichtquelle multipliziert.

Hinweis: Die BRDF eines Lambert'schen Strahlers ist konstant (albedo).

Benötigte Dateien: reflectance.LambertBrdf.java

b) Reflectance Mesh Renderer (2 Punkte)
Implementieren Sie die methode `shade()` des `ReflectanceMeshRenderer`, welche ein Mesh mit mehreren Lichtquellen und mehreren BRDF's zeichnen kann. Rufen sie dazu für jede Lichtquelle und jede BRDF des Materials `getRadiance()` auf und kombinieren sie die Rückgabewerte.

Hinweis: Berechnungen von Richtungen werden im Objektraum vorgenommen.

Benötigte Dateien: renderer.ReflectanceMeshRenderer.java
Mögliche Tests: Reflectance Renderer: Lambert, Reflectance Renderer: Lambert with multiple lightSources

Aufgabe 4 - Cook-Torrance (3 Punkte)

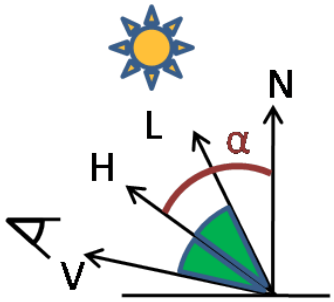
Das Cook-Torrance Beleuchtungsmodell versucht die spekuläre Reflektion realistischer zu modellieren. Das lokale Beleuchtungsmodell modelliert Oberflächen durch viele kleine *Facetten*, welche unterschiedliche Normalenrichtungen haben. Dadurch werden Brechungsverhältnisse, Rauheit und Selbstabschattung berücksichtigt.

Das Cook-Torrance Beleuchtungsmodell verwendet dazu den sogenannten *Half Angle*

Vector H:

$$H = \frac{L+V}{|L+V|}$$

α beschreibt den Winkel zwischen H und der Normalen N.



Das Cook-Torrance Modell besteht hauptsächlich aus drei Teilen:

1. Rauheit:

Es wird angenommen, dass die Oberfläche aus V-förmigen Vertiefungen besteht. Der proportionale Flächenanteil in Richtung α ist gegeben durch den Parameter m, welcher die Rauheit bestimmt:

$$D = \frac{1}{4\,m^2\cos^4(\alpha)}\exp\left(-\frac{1-\cos^2(\alpha)}{\cos^2(\alpha)\,m^2}\right) = \frac{1}{4\,m^2\langle N,H\rangle^4}\exp\left(-\frac{1-\langle N,H\rangle^2}{\langle N,H\rangle^2\,m^2}\right)$$

2. Geometrische Attenuation (Abschwächung)

Die Selbstabschattung wird in drei Fälle unterteilt um den reflektierenden Anteil G zu bestimmen.

$$\begin{aligned} G_1 &= 1 \\ G_2 &= \frac{2\,|\langle N,H\rangle||\langle N,V\rangle|}{|\langle V,H\rangle|} \\ G_3 &= \frac{2\,|\langle N,H\rangle||\langle N,L\rangle|}{|\langle V,H\rangle|} \\ G &= min\{G_1,G_2,G_3\} \end{aligned}$$

3. Fresnel

Dieser Term approximiert den Effekt, dass die Reflektion zunimmt, je weiter die Oberfläche sich von der Blickrichtung wegdreht. (Dieser Effekt kann einfach selbst getestet werden, indem Sie zum Beispiel ein Blatt Papier so zwischen Auge und Lichtquelle halten, dass Sie beinahe nur noch den Rand des Papiers sehen.)

Dies approximieren wir durch:

$$F = r_0 + (1 - r_0)(1 - |\langle H, V \rangle|)^5$$

Die Cook-Torrance BRDF setzt sich schliesslich zusammen zu:

$$I_{out} = albedo \cdot \frac{F \cdot D \cdot G}{4 \langle V, N \rangle}$$

Implementieren Sie das beschriebene Cook-Torrance Modell in der Methode `getRadiance()` der Klasse `CookTorrance`.

Hinweis: Das Skalarprodukt zweier Vektoren der Länge 1 entspricht dem Cosinus des eingeschlossenen Winkels.

Benötigte Dateien: reflectance.CookTorrance.java
Mögliche Tests: Cook-Torrance: Lambert + Cook-Torrance

Aufgabe 5 - Oren-Nayar (3 Punkte)

Das Oren-Nayar Reflektionsmodell versucht im Gegensatz zum Cook-Torrance die diffuse Reflektion realistischer zu modellieren.

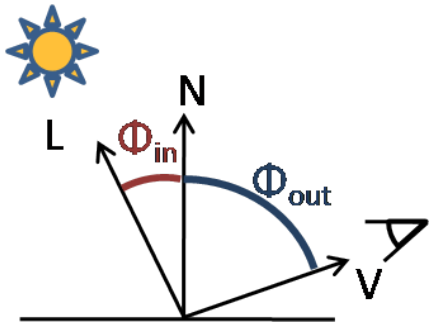
Das Modell bedient sich auch der Vorstellung, dass Oberflächen durch V-förmige Vertiefungen sogenannte Facetten haben. Die Neigungen dieser Vertiefungen werden als Gauss-Verteilt angenommen und haben eine Varianz von σ^2. Im Oren-Nayar Modell ist jede dieser Facetten ein Lambert'scher Strahler.

Durch physikalische Überlegungen gelangt man zur BRDF des Oren-Nayar Reflektionsmodell:

$$I_{out} = albedo \cdot \left[A + \Big(B \cdot max\big(0, \cos(\phi_{in} - \phi_{out})\big) \cdot \sin\alpha \cdot \tan\beta \Big) \right]$$

Wobei:

$$\begin{aligned} A &= 1 - 0.5 \, \frac{\sigma^2}{\sigma^2 + 0.33} \\ B &= 0.45 \, \frac{\sigma^2}{\sigma^2 + 0.09} \\ \alpha &= max(\phi_{in}, \phi_{out}) \\ \beta &= min(\phi_{in}, \phi_{out}) \end{aligned} \tag{1}$$



Implementieren Sie das eben beschriebene Modell für die diffuse Reflektion in der Methode `getRadiance()` der Klasse `OrenNayar`.

Hinweis 1: Das Skalarprodukt zweier Vektoren der Länge 1 entspricht dem Cosinus des eingeschlossenen Winkels.
Hinweis 2: Es ist nicht nötig einen Winkel explizit zu berechnen. Sie können aus Cosinus direkt den Sinus berechnen.
Hinweis 3: Den Tangens haben wir mit $\frac{\sin\beta}{\cos\beta+0.0001}$ berechnet um Divisionen durch null zu vermeiden.

Benötigte Dateien: reflectance.OrenNayar.java
Mögliche Tests: OrenNayar