Hochschule Darmstadt
B. Frömmer

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

04. Dezember 2023

Visual Computing WiSe 2023/24
Aufgabe 4

4.1 Perspektivische Kamera (Wiederholung)

In der ersten Hälfte dieses Aufgabenblattes wollen wir eine Kamera in unser kleines Beispielprogramm integrieren, um eine korrekte perspektivische Projektion in unseren Bildern zu erhalten.

4.1.1 Die ViewMatrix

Bestimmen Sie, wie aus der Vorlesung bekannt, zuerst die `ViewMatrix`, mit derer die komplette Szene so transformiert wird, dass Ihre (virtuelle) Kamera im Ursprung liegt und entlang der negativen Z-Achse schaut. Die Mathe-Bibliothek `glm` stellt bereits eine entsprechende Methode bereit: `glm::lookAt()`. Sie müssen nur noch die Position der Kamera, einen Betrachtungspunkt sowie einen Up-Vektor definieren.

4.1.2 Die ProjectionMatrix

Im zweiten Schritt müssen Sie die perspektivische Projektion der Kamera bestimmen. Diese ist abhängig von den internen Kameraparametern. Die Methode `glm::perspective()` übernimmt die Berechnung der entsprechenden Matrix.

Damit die Kamera beim Rendervorgang berücksichtigt werden kann, müssen Sie die beiden neuen Matrizen noch in den Shader laden und dort korrekt verwenden.

Achtung: Wir hatten im letzten Aufgabenblatt den Tiefentest angeschaltet, mussten hier jedoch noch "falsch herum" testen, da die Invertierung der Z-Achse fehlte (siehe Vorlesung). Das ist nun nicht mehr notwendig, ändern Sie daher Ihren Tiefentest auf folgende Zeilen:

```
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LESS); // Default-Value, koennte auch weggelassen werden
glClearDepth(1.0); // Default-Value, koennte auch weggelassen werden
```

Spielen Sie mit den verschiedenen Parametern der Kamera - was bewirken Änderungen in den Near- und Far-Clipping-Planes, dem (vertikalen) Öffnungswinkel oder dem Seitenverhältnis?

4.1.3 Kamerabewegung (falls noch nicht implementiert)

Implementieren Sie eine rudimentäre Steuerung. Nutzen Sie 'a' und 'd' (X-Achse), 'w' und 's' (Y-Achse) sowie 'q' und 'e' (Z-Achse) zum Bewegen der Kamera (also Verschieben der Kameraposition). Denken Sie daran, den Lookat-Vektor entsprechend zu aktualisieren.

Optional können Sie zusätzlich versuchen, in der onMouseMove()-Methode eine Kamerarotation zu implementieren: Eine Aufwärts- oder Abwärtsbewegung der Maus soll die Kamera nach oben/unten rotieren, eine Seitwärtsbewegung der Maus entsprechend nach links/rechts. Um festzustellen, in welche Richtung sich die Maus bewegt hat, wird nicht nur die aktuelle, sondern auch die alte Mausposition

benötigt. Glücklicherweise enthält das Objekt MousePosition bereits die beide Positionen (in den Feldern X, Y und oldX sowie oldY). Versuchen Sie, über die Methoden der Transform-Klasse die Kamera in die entsprechende Richtung zu rotieren.

Entsprechende Bewegungen (und sonstige Animationen) sind besser in der `update()`-Methode aufgehoben (update wird, vor allem bei forciert niedrigen Framerates, mehrfach pro Frame aufgerufen, was eine stockende Animation verhindert). Hier lassen sich alle Tastatureingaben abfragen und behandeln. Mittels

```
if(m_window->getInput().getKeyState(Key::Up) == KeyState::Pressed)...
```

können Sie beispielsweise überprüfen, ob der Pfeil nach oben gedrückt wurde.

4.2 Beleuchtung

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

In der zweiten Hälfte des Aufgabenblattes werden Sie sich mit der Beleuchtung einer Szene anhand des Phong-Modells beschäftigen. Die obengenannte Gleichung stellt das erweiterte Phong-Modell dar. Zur Auffrischung der Theorie spezifizieren Sie die Buchstaben A - D bzgl. ihres Aufgabenbereiches. In Abbildung 1 sind die verwendeten Winkel visualisiert.

A -

B -

C -

D -

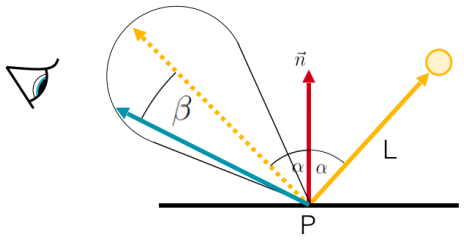

Abbildung 1: Schematische Darstellung des Phong-Modells

4.2.1 Die Punktlichtquelle

Zur Modellierung einer Punktlichtquelle müssen Sie in der `Scene.cpp` folgende Parameter speichern, aktualisieren und pro Frame an die GPU laden:

- Die Position der Lichtuelle (ein 3D-Vector würde hier reichen, Sie können jedoch auch ein Transform-Objekt nutzen und mittels `transform.getPosition()` die Position abfragen)
- Die Lichtfarbe (in RGB, jeweils zwischen 0 und 1)
- Die Lichtintensität (als Faktor zwischen 0 und unendlich)
- Die globale Beleuchtung `ambientLight`
- Die Materialparameter:
  - spekular (typischerweise (1, 1, 1)
  - shininess (typischerweise ein Wert zwischen 20 und 50)

  Später werden diese Parameter eigentlich für jedes Objekt neu bestimmt, aber hier sollen uns Standardwerte reichen. Als diffuses Material können Sie die Farbwerte des Cubes nutzen.

Auch die Lichtquelle soll beweglich gestaltet werden. Nutzen Sie ↑, ↓, ←, → sowie das Komma (Key::Comma) und den Punkt (Key::Period) für entsprechende Bewegungsrichtungen der Punktlichtquelle.

4.2.2 Normalen

Die Licht- und Reflexionsberechnung hängt stark vom Einfallswinkel des Lichtes auf die jeweilige Oberfläche ab. Damit dieser überhaupt bekannt ist, benötigen unsere Objekte vorberechnete Normalen. Tauschen Sie daher den alten `Cube.h` aus durch den erweiterten `Cube_normals.h`, welcher neben Positions- und Farbinformationen auch noch Vertex-Normalen enthält. Denken Sie daran, die `VertexAttribPointer` entsprechend zu aktualisieren und die Normalen im Vertex-Shader entgegenzunehmen. Das Format der neuen Datei sieht dabei wie folgt aus:

$$\underbrace{pos1_x, pos1_y, pos1_z, normal1_x, normal1_y, normal1_z, col1_r, col1_g, col1_b}_{\text{Vertex1}}, pos2_x, pos2_y, pos2_z, normal2_x, ...$$

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 neben dem Vorlesungsskript 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 `toLight` für die Punktlichtquelle im Viewspace und übergeben Sie diese an den Fragment-Shader.

FragmentShader(FS)
Führen Sie die Berechnung der Phong-Komponenten im FS durch. Beachten Sie dabei, dass die Winkel nicht größer als 90° werden dürfen und alle Vektoren vor der Benutzung normalisiert werden müssen.

Sollte alles funktionieren, könnten Sie ein Bild ähnlich diesem erhalten.

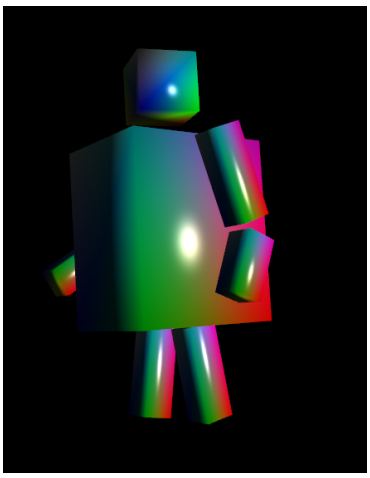

Abbildung 2: Beispielergebnis mit einer Punktlichtquelle