Universität Leipzig
Institut für Informatik
Abteilung für Bild- und Signalverarbeitung
Praktikum Computergrafik

SoSe 2020

Baldwin Nsonga

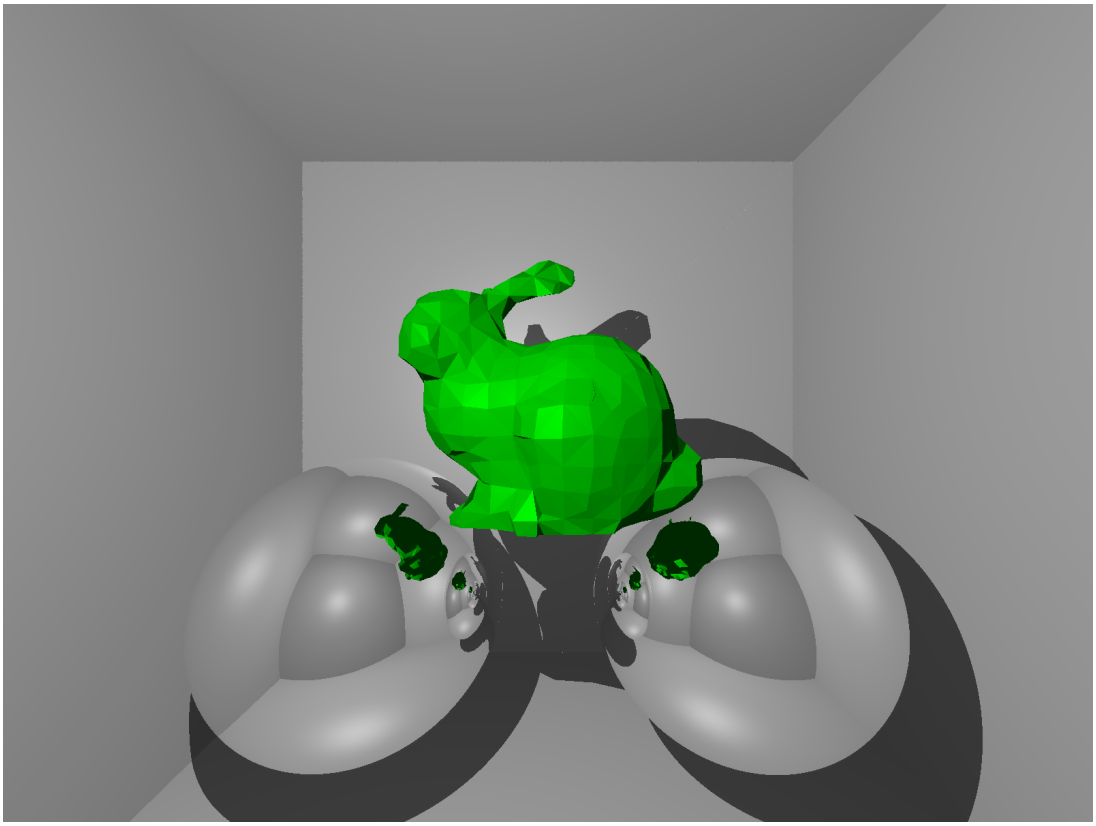

Abbildung 1: Beispielergebnis der Aufgabenserie in hoher Auflösung (1600x1200).

Aufgabenblatt 4

Beleuchtung durch Raytracing

Im Kontext der Beleuchtung wird durch Raytracing berechnet, wie sich Licht innerhalb einer Szene verhält. Hierbei wird zunächst ein Raycasting durchgeführt. An dem Punkt, der durch den Strahl getroffen wird, muss ein lokales Beleuchtungsmodell angewendet werden. Darüber hinaus wird bestimmt, ob sich der Punkt in einem Schatten befindet. Wenn das Material des getroffenen Objektes reflektierend ist, wird ausgehend von Schnittpunkt der reflektierte Strahl berechnet und ein weiterer Schnitttest durchgeführt. Es können ebenso Effekte, wie Lichtbrechnung und Transparenz umgesetzt werden. Diese Prozesse sind rekursiv. In diesem Praktikum begrenzen wir uns auf das Anwenden des Phong-Beleuchtungmodells (Aufgabe 2), Schattierung (Aufgabe 3) und eine "perfekte"Reflexion (Aufgabe 4) mit nur **einer** Punktlichtquelle.

Informieren Sie sich selbstständig zu Beleuchtungmodellen (Inhalt der Vorlesung am 9. Juni) und Raytracing (Inhalt der Vorlesung am 16. Juni).

Diese Aufgabenserie setzt voraus, dass Aufgabenblatt 3 vollständig gelöst wurde. Bitte

bereiten Sie für die Abnahme mindestens ein Bild mit den Beispieltransformationen und Beispielparametern vor. Alternativ können Sie unter folgenden Vorraussetzungen auch eine persönliche Szene präsentieren: Phong Beleuchtungsmodell muss klar erkennbar sein, Schatten müssen klar erkennbar sein, **rekursive** Reflexion muss erkennbar sein.

Aufgabe 1 - Vorbereitungen / Raycasting

Folgendes muss in dieser Aufgabe implementiert werden:

1. *main(...)*: Fügen Sie der *Scene* mit *Scene::addPointLight* eine Punktlichtquelle hinzu. In den Beispielen entspricht die Position der Punktlichtquelle der Position der Kamera mit einer Verschiebung um [-100, 100, -100].

2. *main(...)*: Die Kugeln wurden in den Beispielen modifiziert: Kugel 1: Position = [-200, -200, -150], Radius = 150; Kugel 2: Position = [200, -200, -150], Radius = 150.

3. *main(...)*: Das Bunny-Modell wurde 10 Einheiten in negativer y Richtung und 30 Einheiten in negativer z Richtung verschoben und um 170 Grad **um** die y-Achse rotiert (bitte beachten Sie, dass das Modell um die eigene Achse gedreht wurde). Darüber hinaus wurde es um einen Faktor von 1.5 in alle Raumrichtungen skaliert.

4. *main(...)*: Das Cube-Modell wurde 10 Einheiten in positiver y Richtung verschoben. Darüber hinaus wurde es um einen Faktor von 10 in alle Raumrichtungen skaliert. Es umhüllt damit die Kugeln und das Bunny Modell und soll damit unseren Raum darstellen. Da die Normalen des Cube-Modells nach außen gerichtet sind, wir uns aber im Inneren der Würfels befinden, werden in den Beispielen die Normalen des Cube-Modells umgekehrt.

5. *main(...)*: Erzeugen Sie drei Objekte der Klasse *Material*. Ein Material für das Bunny-Modell, ein Material für das Cube-Modell und ein weiteres Material für die Kugeln. Setzen Sie die Farben des Materials und fügen Sie das Material mit Hilfe von *setMaterial* einem Modell/Kugel hinzu. (Bunny: [0,1,0], Cube: [0.8,0.8,0.8], Kugeln: [0,0,1]). Darüber hinaus wird in Aufgabe3 eine "perfekte"Reflexion auf den Kugeln umgesetzt . Setzen Sie entsprechend die *reflection* des Kugelmaterials auf 1.0.

6. Erweitern Sie Ihre Implementierung des Raycasting, sodass bei einem Treffer die Materialeigenschaften des getroffenen Objektes mit der kürzesten Distanz zur Kamera ausgelesen werden und die Pixelfarbe entsprechend dieser Eigenschaften gesetzt wird. Dies soll umgesetzt werden, indem bei einer positiven Evaluierung der Funktion *Scene::intersect(...)* innerhalb der Funktion *SolidRenderer::computeImageRow(size_t rowNumber)* die Funktion *SolidRenderer::shade(HitRecord &r)* ausgeführt wird. In der Funktion *SolidRenderer::shade(HitRecord &r)* wird geprüft ob im *HitRecord* ein Treffer mit einem Objekt verzeichnet ist. Ist dies der Fall, so wird aus dem Hitrecord

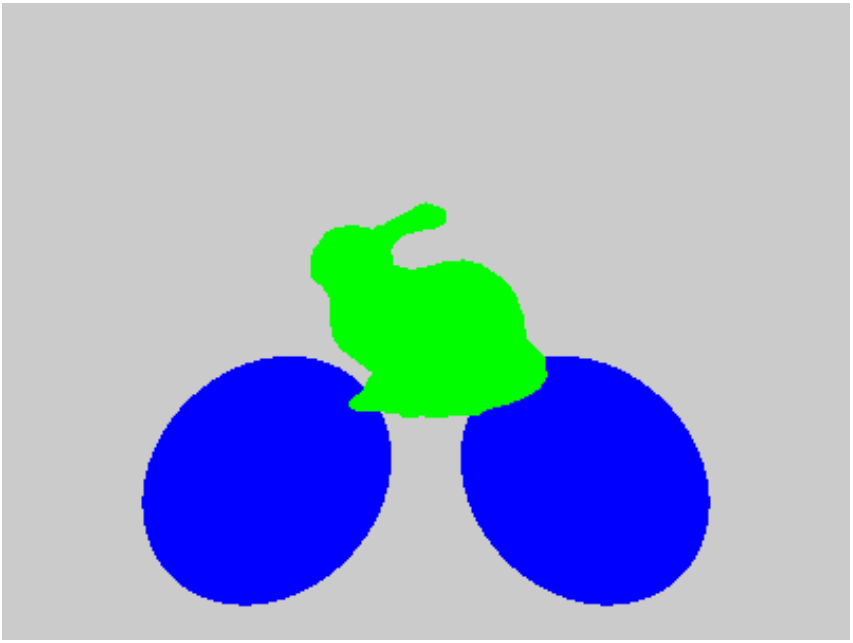

Abbildung 2: Zielsetzung der Aufgabe 1

die Materialeigenschaften ausgelesen. In shade wird dann die *color* des *HitRecord* geschrieben. *Image::setValue(...)* wird in *SolidRenderer::computeImageRow(size_t rowNumber)* mit der aktualisierten *HitRecord* Farbe ausgeführt. Beispielergebnis: Abbildung 2.

Aufgabe 2 - Lokales Beleuchtungsmodell: Phong

Folgendes muss in dieser Aufgabe implementiert werden:

1. Erweitern Sie Ihre Schnittpunkttests so, dass die Informationen zum Schnittpunkt, der Normalen und der aktuellen Strahlenrichtung im *HitRecord* festgehalten werden. Im folgenden wird für Modelle das **FLAT**-shading umgesetzt. Kugeln werden durch **PHONG**-shading berechnet. Das bedeutet, dass Sie für die in den Dreiecken gespeicherten Normalen (nach der Transformation) direkt für die Berechnung des Beleuchtungsmodells nutzen können. Für die Kugeln muss für jeden Schnittpunkt eine Normale berechnet werden.

2. Erweitern Sie die Funktion *SolidRenderer::shade(HitRecord &r)*, die bisher lediglich die Farbe des getroffenen Materials ausliest und im HitRecord speichert so, dass das Phong-Beleuchtungsmodell umgesetzt wird. Die Berechnung für nur eine Lichtquelle umgesetzt werden. Zur Vereinfachung entspricht die Lichtintensität einem weißen Licht mit voller Intensität für alle Anteile ([1,1,1]). Die die entsprechende Intensität

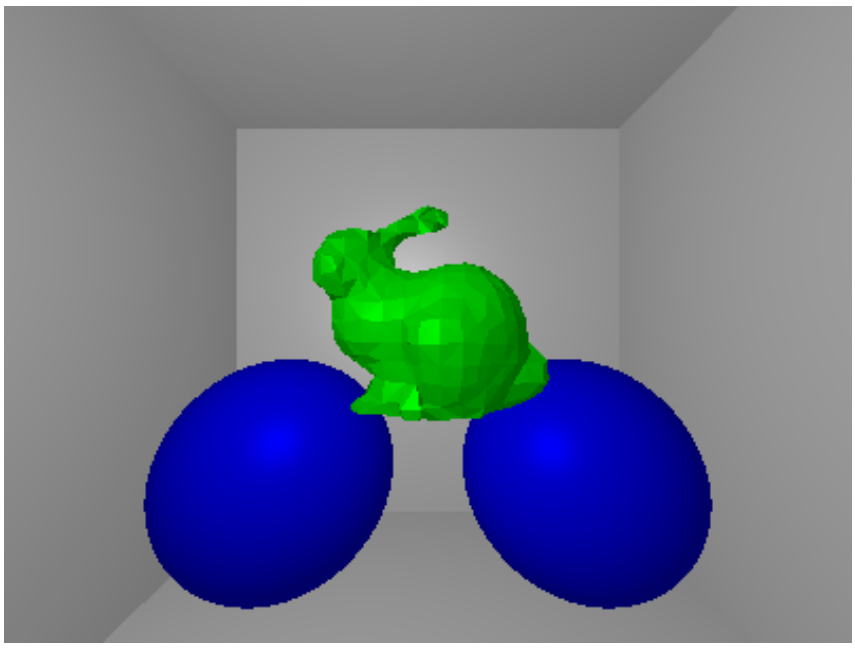

Abbildung 3: Zielsetzung der Aufgabe 2

kann in einem einzigen *double* mit Wertebereich $[0, 1]$ festgehalten werden, welcher mit der Materialfarbe multipliziert wird. Außerdem wurden in den Beispielen die Parameter $k_{\text{ambient}} = 0.4$, $k_{\text{diffus}} = 0.4$ und $k_{\text{specular}} = 0.2$ gewählt. Für die Berechnung des Glanzlichtes (specular) wurde eine $n = 10$ Potenz gewählt. Beispielergebnis: Abbildung 3.

Aufgabe 3 - Schatten

Nach dem Berechnen der lokalen Beleuchtung muss entschieden werden, ob der Punkt im Schatten liegt. **Erweitern Sie die Funktion** *SolidRenderer::shade(HitRecord &r)*:

1. Berechnen Sie ausgehend vom Schnittpunkt einen normalisierten Schattenstrahl. Dieser führt vom Schnittpunkt zur Lichtquelle. Beachten Sie hierbei, dass das Objekt sich nicht durch numierische Ungenauigkeiten selbst schneidet. Dies unterbinden Sie, indem Sie den Ursprung des Schattenstrahles leicht verschieben.

2. Erzeugen Sie einen HitRecord für die Schattenberechnung. Beachten Sie bei der Initialisierung die Variable *parameter* (Schnitte sind nur relevant, wenn sich das Objekt zwischen dem Punkt und der Lichtquelle befindet).

3. Führen Sie die Funktion *Scene::intersect(...)* mit dem Schattenstrahl und dem HitRecord für die Schattierung durch. Wird ein Objekt getroffen, das zwischen dem

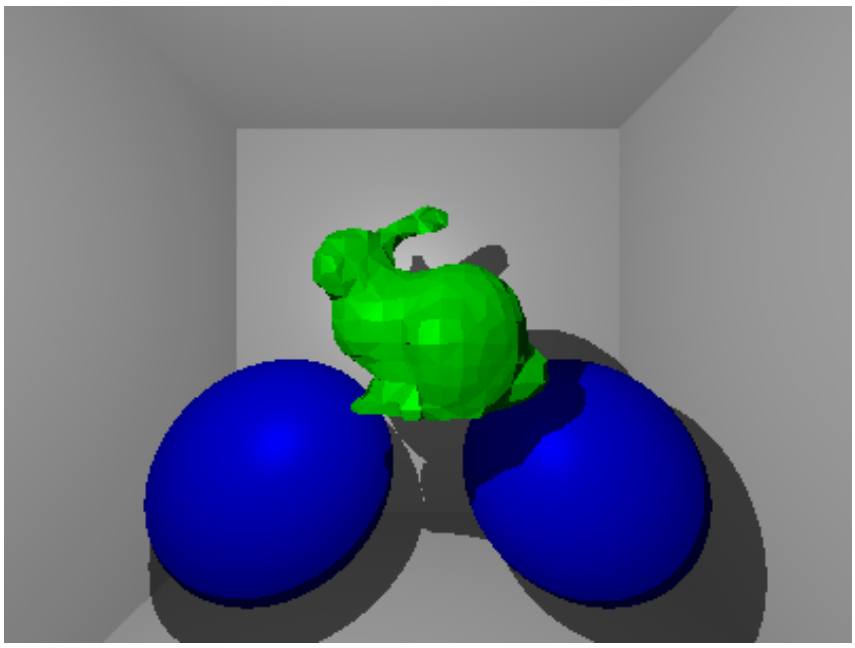

Abbildung 4: Zielsetzung der Aufgabe 3

Punkt und der Lichtquelle liegt, so liegt der Schnittpunkt im Schatten. Ist dies der Fall, muss die intensität der Farbe des globalen HitRecord reduziert werden. (Im Beispiel entspricht die Farbe der Ambienten intensität: r.color *= 0.4). Beispielergebnis: Abbildung 4

Aufgabe 4 - Reflexion

Wird ein reflektierendes Material getroffen, muss ein Reflexionsstrahl erzeugt und ausgewertet werden. Im Rahmen dieses Praktikums können Materialien entweder vollständig ("perfekt") oder nicht reflektieren. **Erweitern Sie die Funktion** *SolidRenderer::shade(HitRecord &r)*:

1. Modifizieren Sie die Materialeigenschaften der Kugel so, dass Sie vollständig reflektieren (reflection = 1.0).

2. Ist das Material reflektierend (reflection > 0.0), müssen Sie einen Reflexionsstrahl berechnen und im Fall einer perfekten Reflektion den aktuellen Farbwert auf des HitRecord auf [0,0,0] setzen. Beachten Sie, dass der Reflexionsstrahl nicht durch numerische Ungenauigkeiten das getroffene Objekt selbst schneidet.

3. Aktualisieren Sie den globalen HitRecord, so dass getroffene Modelle, Dreiecke und Kugeln den default Wert haben. Darüber hinaus muss *parameter* neu initialisiert

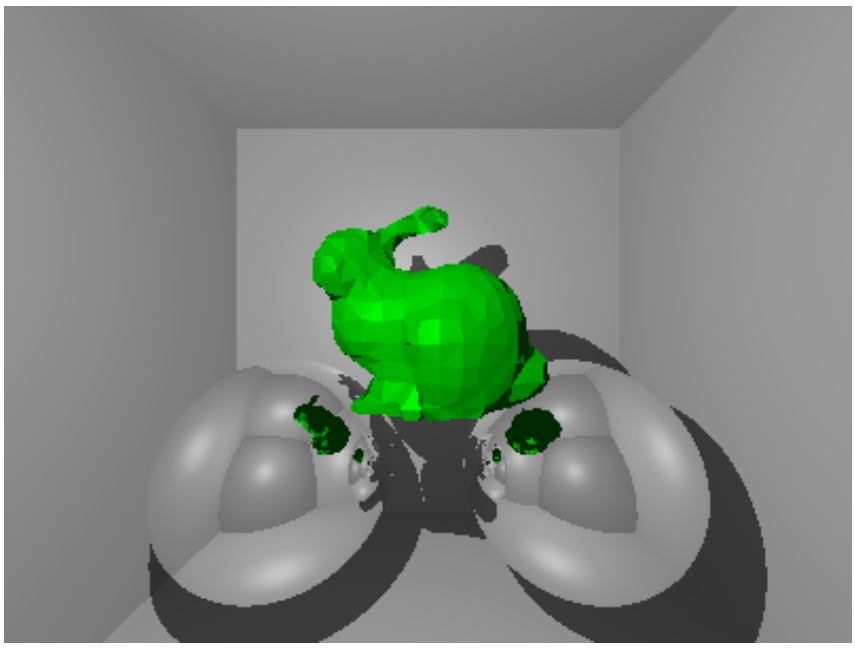

Abbildung 5: Zielsetzung der Aufgabe 4. Entspricht Abbildung 1 in geringerer Auflösung.

werden. Beachten Sie die Variable *recursions* des Hitrecord. Diese muss die aktuelle Rekursionstiefe aufzeichnen. In dem Fall, dass die vordefinierte Rekursionstiefe erreicht ist, muss die Berechnung der Reflexion abbrechen. Im Beispiel wurde eine maximale Rekursionstiefe von 5 gewählt.

4. Führen Sie die Funktion *Scene::intersect(...)* mit dem Reflexionsstrahl und dem aktualisierten globalen HitRecord durch. Wird ein Objekt getroffen, muss *SolidRenderer::shade(HitRecord &r)* rekursiv aufgerufen werden. Beispielergebnis: Abbildung 5