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

SoSe 2024

Baldwin Nsonga
Lydia Müller

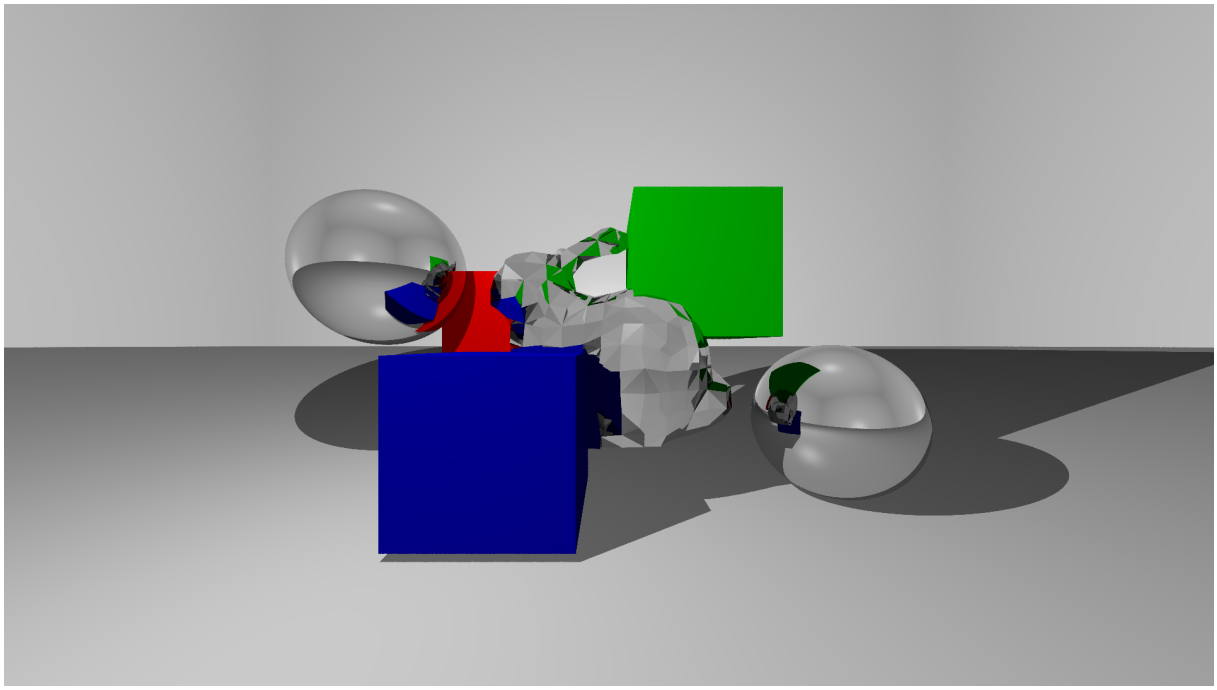

Abbildung 1: Beispielergebnis in Auflösung 1920x1080 und 5x Antialiasing. Kugeln wurden um 50 in positive und negative y-Richtung verschoben.

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 vom 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 1), Schattierung (Aufgabe 2) und eine "perfekte"Reflexion (Aufgabe 3) mit nur **einer** Punktlichtquelle und einem Antialiasing. Um die Effekte besser sichtbar zu machen, wird ein "Raum"benötigt. Fügen Sie zur Vorbereitung der Szene einen Würfel hinzu, der die kleinen Würfel sowie den Hasen umschließt. Achten Sie darauf, dass die Oberflächennormen im Würfel nach außen zeigen.

Informieren Sie sich selbstständig zu Beleuchtungmodellen und Raytracing (Inhalte der Vorlesung).

Diese Aufgabenserie setzt voraus, dass das Aufgabenblatt 3 vollständig gelöst wur-

de. 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 mit eigenen Modellen und individuellen Parametern präsentieren: das Phong Beleuchtungsmodell, Schatten sowie Reflexionen mit Rekursion müssen klar erkennbar sein.

Aufgabe 1 - 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 (Es kann auch Phongshading implementiert werden). 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 soll für nur eine Lichtquelle umgesetzt werden. Zur Vereinfachung entspricht die Lichtintensität einem weißen Licht mit voller Intensität. In den Beispielen wurden 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 $n = 20$ Potenz gewählt. Hinweis: Die Behandlung von Vorder- und Rückseiten der Flächen steht Ihnen frei. In dem Beispiel werden die Vor- und Rückseite gleich beleuchtet.

Aufgabe 2 - 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

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: intensity $* = 0.5$).

Aufgabe 3 - 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 des Hasen und der Kugeln so, dass sie vollständig reflektiert (reflection = 1.0). Die Materialfarbe wird auf **schwarz** gesetzt.

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 entsprechend angepasst werden. 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 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.

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.

Aufgabe 4 - Antialiasing

Hier werden pro Pixel mehrere Strahlen erzeugt und das Ergebnis gemittelt. Erweitern Sie die Methode *getRay(...)* der Klasse Camera so, dass der Strahl zufällig oder strukturiert in x und y Richtung variiert wird.

```
double jitterx = ( drand48() - 0.5 ) / (double)mWidth;
double jittery = ( drand48() - 0.5 ) / (double)mHeight;
```

Abbildung 2: Mögliche Implementierung für "jitter"(zufällige Variation)

```
r.direction = r.direction + mViewTransform.getColumn( i: 0 ) * jitterx + mViewTransform.getColumn( i: 1 ) * jittery;
```

Abbildung 3: Mögliche Anpassung des Strahls

Implementieren Sie dann das Antialiasing mit den modifizierten Strahlen im Solid-Renderer. Experimentieren Sie mit angemessenen Parameters für die Bildauflösung und der Strahlenanzahl pro Pixel.