# Aufgabe Raytracing

# 5 Praxisaufgaben zum Raytracing

## 5.1 Einstieg in das Projekt

Der Übung liegen verschiedene Materialien bei. Dabei handelt es sich um eine Musterlösung (für Windows), Quelltexte, eine Solution für Visual Studio 2012, CMake und eine OBJ-Datei mit einem Dreiecksnetz. Die Quellcodes der Musterlösung finden Sie in dem Ordner `src_solution`.

Die Quelltextstellen, welche von Ihnen in den einzelnen Aufgaben bearbeitet werden sollen, sind mit den Kommentaren `//student begin` bzw. `//student end` markiert.

Im Rahmen dieser Aufgabe sollen verschiedene Teile eines vorgegebenen Raytracers vervollständigt werden. Es handelt sich um ein Konsolenprogramm, was als Ergebnis ein BMP-Bild schreibt.

Machen Sie sich zunächst ein wenig überblicksmäßig mit der Struktur des Raytracers vertraut.

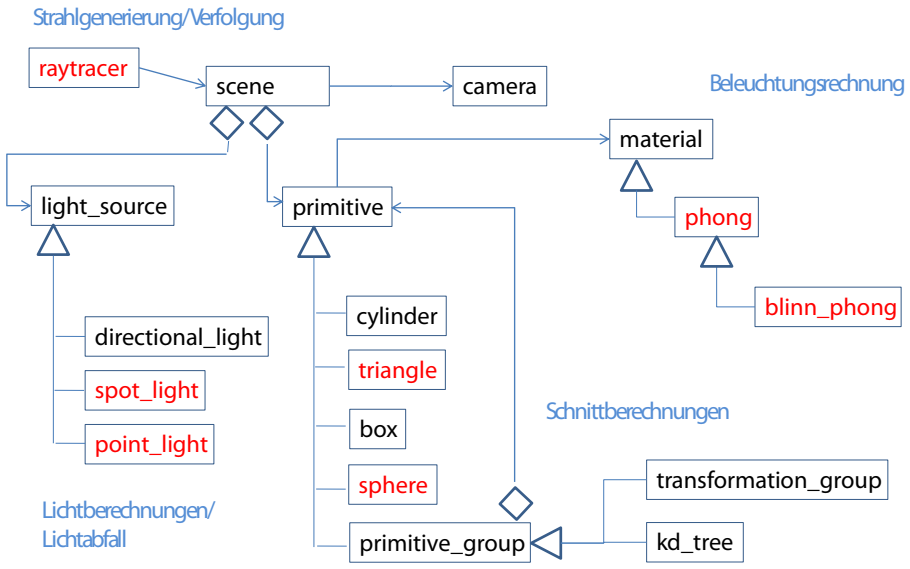


Abbildung 1: Raytracer: Klassenübersicht

Zu Beginn sollte der Raytracer Bild 2 ausgeben.

## 5.2 Hinweise zum schnelleren Testen

Da es sich bei dem Raytracer um ein recht komplexes Programm handelt, kann die Ausführung einige Zeit in Anspruch nehmen. Um diese Zeit zu verkürzen, können Sie folgende Schritte probieren:

- Reduzieren Sie die Auflösung des generierten Bilds. Diese wird in `main.cpp` durch die Zuweisung der beiden Variablen `resx` und `resy` festgelegt.
- Kompilieren Sie das Projekt im Release-Modus. Eine Auswahlliste dazu befindet sich üblicherweise am oberen Rand von Visual Studio. Beachten Sie dabei aber, dass die Suche von Pro-

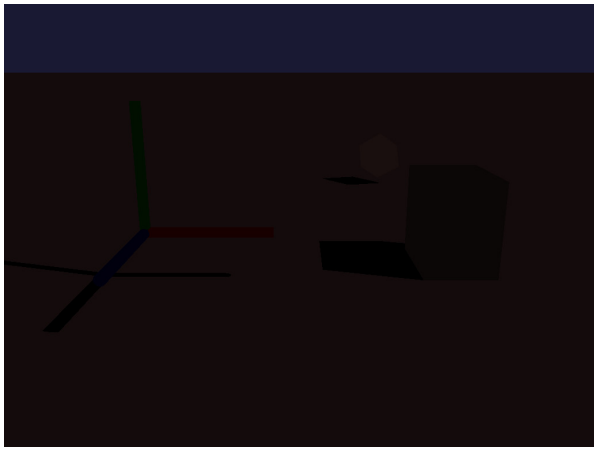

Abbildung 2: Ausgangszustand

grammfehlern dabei sehr erschwert wird, da nicht alle dafür benötigten Informationen generiert werden. Sind in Ihrem Programm also noch Fehler, belassen Sie die Einstellung lieber auf `Debug`.

- Starten Sie das Programm "Ohne Debuggen" mit Strg+F5.

Der Raytracer ist mittels OpenMP für eine parallele Ausführung konzipiert. Wenn Sie einen Compiler verwenden, der OpenMP nicht standardmäßig integriert hat (bspw. die Express-Versionen von Visual Studio), müssen sie den OpenMP-Header `#include <omp.h>` in `raytracer.cpp` auskommentieren.

## 5.3 Schnittberechnungen

Alle Objekte der Szene sind von der Klasse `primitive` abgeleitet und müssen zwei Schnittfunktionen implementieren: `closest_intersection` und `any_intersection`. Die Funktion `any_intersection` wird für den Schattentest benutzt und bekommt als Parameter den Strahl `r`, die Intervallgrenzen `min_lambda` und `max_lambda` sowie den Zeiger auf ein Primitiv namens `dont_hit`. Die Funktion muss `true` zurückliefern, wenn das Objekt vom Strahl geschnitten wird, der Schnittpunkt im Intervall zwischen `min_lambda` und `max_lambda` und das Objekt nicht identisch mit `dont_hit` ist. Die Schnittposition selbst muss nicht unbedingt ausgerechnet werden. Es reicht eine `true` oder `false` Entscheidung.
Die Routine `closest_intersection` bekommt als Parameter einen Zeiger auf die `intersection_info` namens `hit`, die untere Intervallgrenze `min_lambda` und den Primitivezeiger `dont_hit`. Die Routine liefert `false` wenn der Zeiger `dont_hit` auf das aktuelle Objekt zeigt oder kein Schnittpunkt gefunden wurde, der weiter weg als `min_lambda` und näher dran als `intersection_info->get_lambda()` liegt. Zu berechnen ist der Schnitt mit dem Strahl, der von `get_incoming_ray` zurückgegeben wird. Gibt es einen solchen Schnittpunkt, so aktualisiert die Routine die Daten in der Schnittinformation `hit` und liefert `true` zurück. Dabei müssen drei Informationen aktualisiert werden. Der Zeiger auf das aktuelle Primitive mit der Hilfsfunktion `hit->set_object`, die Oberflächennormale am Schnittpunkt mittels `hit->set_normal` und die neue Entfernung des Schnittpunktes mit `hit->set_lambda`.
Orientieren Sie sich an den bereits fertigen Implementierungen von `plane`, `box` oder `cylinder`.

Die Strahlen und Vektoren der `intersection_info`-Struktur sind wie folgt ausgerichtet:

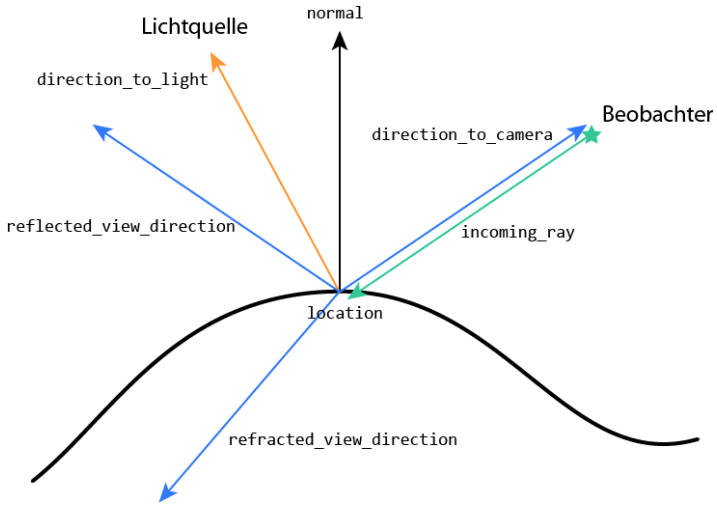


Abbildung 3: Raytracer: intersection_info

### 5.3.1 Strahl-Kugel-Schnitt

Implementieren Sie die Methoden `closest_intersection` und `any_intersection` der Klasse `sphere`. Schauen Sie sich dazu im Skript die Berechnung des Strahl-Kugel-Schnittes auf Folie 13 an. Tipp: Im Header `utils.h` gibt es eine Hilfsroutine namens `solve_real_quadratic`, die eine quadratische Gleichung der Form $ax^2+bx+c=0$ löst. Geben Sie dieser Funktion die drei Koeffizienten der Gleichung und ein zweidimensionales Array, in das die Funktion die Lösungen schreiben kann. Die Funktion gibt die Anzahl der Lösungen zurück.

Abbildung 4 zeigt, wie das Ergebnis nach diesem Schritt aussehen soll.

Abbildung 4: Zwischenstand, Strahl-Kugel-Schnitt

### 5.3.2 Strahl-Dreiecks-Schnitt

Implementieren Sie die Methoden `closest_intersection` und `any_intersection` der Klasse `triangle`.

Abbildung 5 zeigt, wie das Ergebnis nach diesem Schritt aussehen soll.

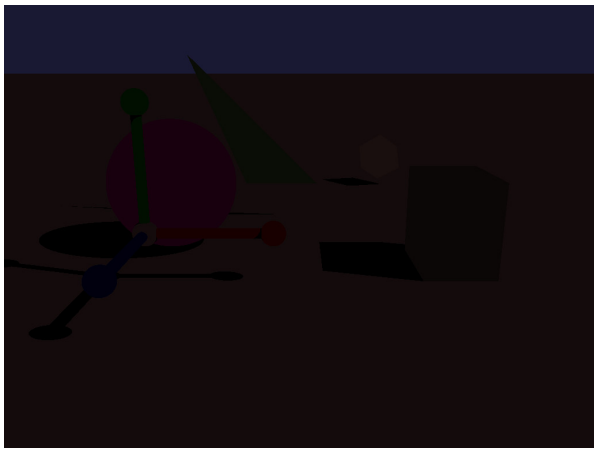

Abbildung 5: Zwischenstand, Strahl-Dreiecks-Schnitt

## 5.4 Beleuchtungsmodelle

Die Beleuchtungsberechnung erfolgt beim Material. Bereits vorgesehene Implementierungen sind die Klasse `phong` und `blinn_phong`. Die Beleuchtung wird in drei Teile zerlegt: das ambiente Umgebungslicht, die ungerichtete diffuse Reflexion und die gerichtete spekulare Reflexion. Die `material`-Klasse ist dafür verantwortlich, die Lichtkomponenten zu aggregieren. Die Berechnungen der einzelnen Komponenten werden von den abstrakten Methoden `shade_diffuse` und `shade_specular` ausgeführt und müssen in konkreten Unterklassen implementiert werden. Die Berechnung des Umgebungslichts ist bereits vorimplementiert. Die Klasse `phong` ist von `material` abgeleitet und überschreibt die beiden genannten Funktionen. `blinn_phong` unterscheidet sich vom Phong-Modell nur in der Berechnung des spekularen Anteils. Daher erbt `blinn_phong` von `phong` und überschreibt nur `shade_specular`. Die Berechnungsmethoden für die Lichtkomponenten bekommen die Schnittinformation `hit` und einen Zeiger auf die zu verwendende Lichtquelle. Als Rückgabewert soll die lokal berechnete Farbe der entsprechenden Komponente als dreidimensionaler Vektor zurückgeliefert werden. Die Methode `shade` und damit auch `shade_diffuse` und `shade_specular` werden vom Raytracer nur an Oberflächenpunkten aufgerufen, die nicht im Schatten liegen. Der Schattentest wurde bereits in der Methode `trace` des Raytracers durchgeführt.

Die Abschwächungskoeffizienten des Lichts müssen mit in die Beleuchtungsberechnung eingehen. Diese geben an, welcher Bruchteil der Intensität der Lichtquelle an dem aktuellen Oberflächenpunkt noch ankommt. Sie sind z.B. abhängig von der Entfernung. Die Koeffizienten können mit der Methode `get_attenuation_factors` der Lichtquelle abgerufen werden. Bei dem Rückgabewert handelt es sich um einen dreidimensionalen Vektor, der die Abschwächung für die drei Farbkanäle angibt. Eine 1 bedeutet dabei, dass das Licht gar nicht abgeschwächt wird und in voller Intensität auf den Oberflächenpunkt trifft, eine 0 bedeutet, dass gar kein Licht mehr ankommt.

### 5.4.1 Phong

Implementieren Sie die Berechnung des diffusen und des spekularen Anteils nach dem Phong-Modell in den Methoden `shade_diffuse` und `shade_specular} der Klasse \verb`phong|. Die Materialparameter stellt die Elternklasse `material` bereit. Die Oberflächennormale, die Lichtrichtung, der einfal-

lende Strahl (vom Betrachter) und einiges mehr stehen über die Schnittinformation `hit` zur Verfügung. Die Lichtfarbe und die Lichtabschwächung erhält man von der übergebenen Lichtquelle.

Abbildung 6 zeigt, wie das Ergebnis nach diesem Schritt aussehen soll.

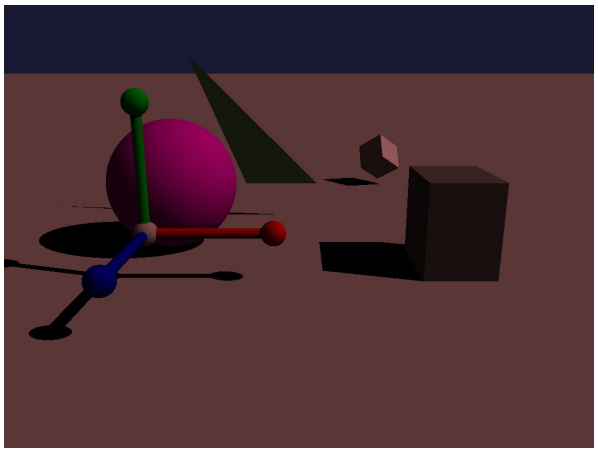

Abbildung 6: Zwischenstand, Phong-Beleuchtungsmodell

### 5.4.2 Blinn-Phong

Implementieren Sie die Berechnung des spekularen Anteils nach dem Blinn-Phong-Modell in der Methode `shade_specular` der Klasse `blinn_phong`.

Abbildung 7 zeigt, wie das Ergebnis nach diesem Schritt aussehen soll.

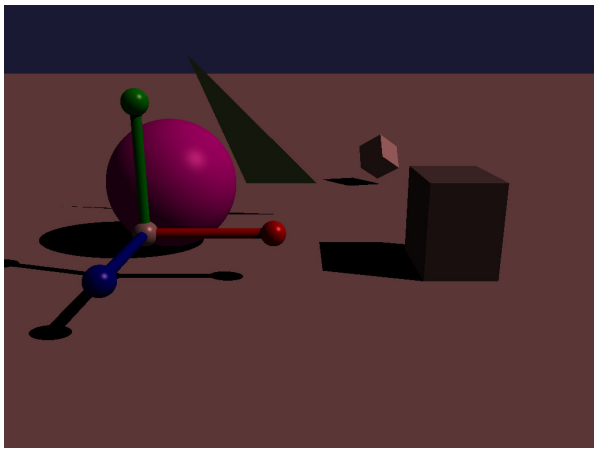

Abbildung 7: Zwischenstand, Blinn-Phong-Beleuchtungsmodell

## 5.5 Erweiterung der Grundszene

Testen Sie ihre Implementierung in dem Sie zwei weitere Beispielszenen erzeugen in dem ihre Primitive und ihre beiden Materialen benutzt werden. Orientieren Sie sich bei der Erstellung der Szene an der

bereits fertigen Testszene `create_scene` in der Datei `main.cxx`. Vergessen Sie nicht die neuen Szenen dem Raytracer in der `main`-Funktion zu übergeben.

## 5.6 Lichtquellen

Alle Lichtquellen sind von der Klasse `light_source` abgeleitet und implementieren die zwei Methoden `get_attenuation_factors` und `calc_light_direction_and_distance`. Die zweite Methode namens `calc_light_direction_and_distance` bekommt die bereits ausgefüllte Schnittinformation `hit` übergeben und ergänzt noch die Distanz und Richtung zur Lichtquelle mit den Hilfsroutinen `hit->set_light_dir` und `hit->set_light_distance`.

Dafür benötigt sie meist noch die Position des Schnittpunktes, welche mittels `hit->get_location` erfragt werden kann. Die Routine `get_attenuation_factors` liefert den Abschwächungsfaktor der Lichtquelle pro Farbkanal (als Vektor) zurück. Beim gerichteten Licht ist der Abschwächungsfaktor einfach der Vektor $(1, 1, 1)^T$. Das gerichtete Licht ist bereits in der Klasse `directional_light` fertig als Beispiel fertig implementiert.

Bei einem Punktlicht berechnet sich die Abschwächung nach folgender Formel:

$$a_{point}(r) = \frac{1}{c + l \cdot r + q \cdot r^2} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

wobei $r$ der Abstand zwischen Oberflächenpunkt und Lichtquellenposition und $c$ der konstante, $l$ der lineare und $q$ der quadratische einstellbare Abschwächungskoeffizient der Lichtquelle sind.

Bei einem Spot gibt es zusätzlich noch den `spot_exponent` $n$, die normierte Hauptrichtung des Strahlers `spot_direction` $\vec{d}$ und den Kosinus des Cutoff-Winkels `spot_cos_cutoff`. Ist der Winkel zwischen negiertem Lichtvektor $\vec{L}$ und der Hauptrichtung des Strahlers kleiner als der Kosinus des Cutoff-Winkels. So ist die Abschwächung

$$a_{spot} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

.

Ansonsten berechnet sich die Abschwächung nach folgender Formel

$$a_{spot} = \langle -L, d \rangle^n \cdot \frac{1}{c + l \cdot r + q \cdot r^2} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

### 5.6.1 Punktlichtquelle

Implementieren Sie zwei Methoden `get_attenuation_factors` und `calc_light_direction_and_distance` für eine Punktlichtquelle in der Klasse `point_light`.

Abbildung 8 zeigt, wie das Ergebnis nach diesem Schritt aussehen soll.

### 5.6.2 Strahler/Spot

Implementieren Sie zwei Methoden `get_attenuation_factors` und `calc_light_direction_and_distance` für eine Spot in der Klasse `spot_light`.

Abbildung 9 zeigt, wie das Ergebnis nach diesem Schritt aussehen soll.

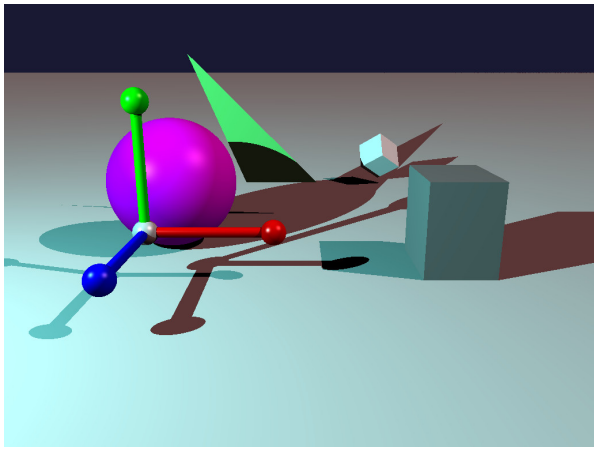

Abbildung 8: Zwischenstand, Punktlichtquelle

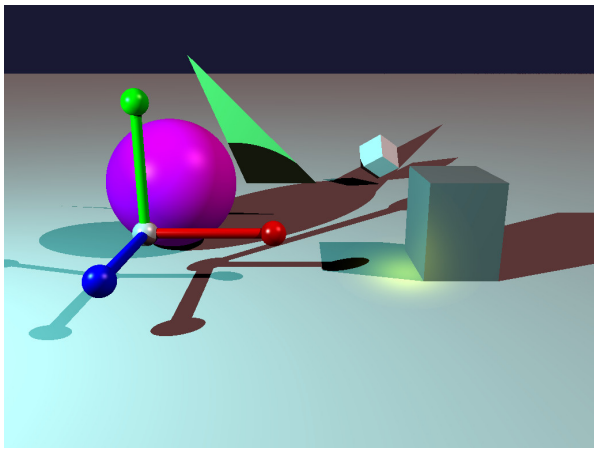

Abbildung 9: Zwischenstand, Spot

## 5.7 Rekursives Raytracing

In der Methode `render` der Klasse `raytracer` befindet sich die Hauptschleife des Raytracers, welche über alle Pixel des Bildes läuft und ein oder mehrere Strahlen generiert, welche dann mit der Methode `trace` in die Szene verfolgt werden. Schauen Sie sich zunüchst diese Methode genauer an. In der momentanen Implementierung berechnet die Routine `trace` den nächsten Schnittpunkt mit der Szene, und iteriert dann über alle Lichtquellen. Befindet sich eine Lichtquelle nicht im Schatten oder ist bei ihr die Schattengenerierung deaktiviert, so wird für den gefundenen Schnittpunkt die Methode `shade` des zugehörigen Materials mit der Lichtquelle aufgerufen. Die so berechneten Farben werden akkumuliert und von der Routine `trace` zurückgeliefert.

### 5.7.1 Rekursion für die Reflexion

Erweitern Sie die Routine `trace` um einen rekursiven Aufruf für die Reflektion. Verwenden sie die übergebene aktuelle Rekursionstiefe und die als Attribut im Raytracer verfügbare maximale Rekursionstiefe als Abbruch Kriterium. Generieren Sie analog zum Primärstrahl eine neue Schnittinformation für die Verfolgung des reflektierten Strahls. Verwenden Sie die Materialkonstante der Reflektivität,

welche mittels `get_reflectivity` des aktuellen Objektmaterials erfragbar ist, um den Farbbeitrag des reflektieren Strahls dazuzurechnen.

Abbildung 10 zeigt, wie das Ergebnis nach diesem Schritt aussehen soll.

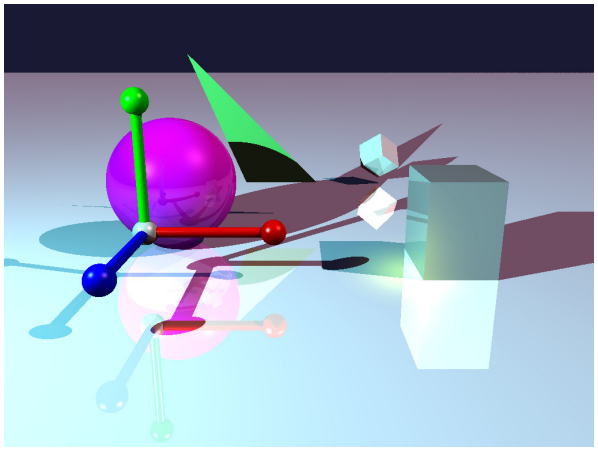

Abbildung 10: Zwischenstand, Reflexion

## 5.8 Zusatzaufgaben

*Hinweis: Die erreichbare Bonuspunktezahl für eine korrekt gelöste Zusatzaufgabe finden Sie am Ende der jeweiligen Aufgabenbeschreibung. Maximal können jedoch nicht mehr als 3 Bonuspunkte von diesem Übungsblatt eingebracht werden.*

- Realisieren Sie einen Kegel als eigenes Primitive. (0,5 Pkt.)
- Realisieren Sie einen Torus als eigenes Primitive. (0,5 Pkt.)
- Erweitern Sie den Raytracer um eine flächige Lichtquelle. (2 Pkt.)
- Erweitern Sie den Raytracer um die Brechung. (1,5 Pkt.)
- Die momentane Implementierung des Raytracers parallelisiert die Berechnung mittels OpenMP auf Zeilenbasis. Stellen Sie dies um auf Kachelbasis mit einstellbarer Kachelgröße. Prüfen Sie ob sich dadurch ein Geschwindigkeitsvorteil ergibt? (3 Pkt.)
- Der momentan verwendete kd-Baum basiert auf der Median-Split Heuristik. Stellen Sie die Generierung auf die SAH-Heuristik um und vergleichen Sie die Performance. (3 Pkt.)
- Erweitern Sie den Raytracer um ein Szenenbeschreibungsformat. (3 Pkt.)