Karlsruher Institut für Technologie
Lehrstuhl für Computergrafik

Prof. Dr.-Ing. Carsten Dachsbacher
Dipl.-Inform. Johannes Meng, Dipl.-Inform. Florian Simon,
M.Sc. Emanuel Schrade

2. Übungsblatt zur Vorlesung Computergraphik im WS 2016/17

Abgabe bis Montag, 28.11.2016, 11:00 Uhr.

1 *Whitted-Style Raytracing* — 20 Punkte

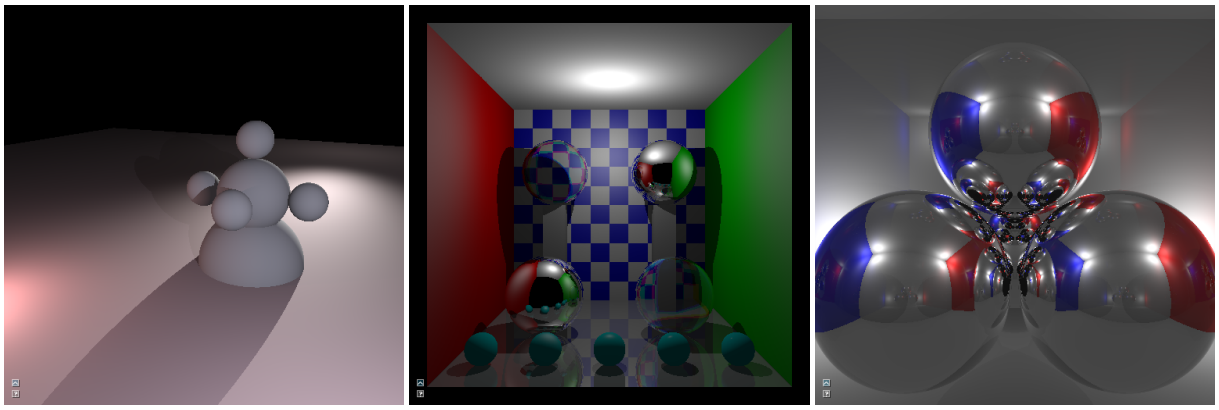


Abbildung 1: Ergebnisbilder des Übungsblattes und eine Strahl-Schnitt Visualisierung.

In diesem Übungsblatt sollen Sie Teile eines Raytracing-Algorithmus implementieren. Dieser erweitert das Whitted-Style-Raytracing, das Sie in der Vorlesung kennen gelernt haben (Phong-Beleuchtungsmodell, Rekursive Verfolgung von Reflexion und Transmission), um die Behandlung von Fresnel-Effekt und Dispersion. Das Resultat einer kompletten Implementierung ist in Abbildung 1 dargestellt. Im Folgenden werden wichtige Teile des Frameworks beschrieben. Lesen Sie sich bitte die Beschreibung sorgfältig durch und machen Sie sich mit dem Code des Frameworks vertraut.

- Der Programmablauf startet mit der `main`-Funktion in der Datei `main.cpp`. Hier wird ein Objekt der Klasse `RaytracingContext` erstellt, das alle für das Raytracing relevanten Informationen enthält. Dazu gehören z.B. die Kamera, die Objekte in der Szene und die Parameter, die für die Bilderzeugung verwendet werden sollen. Danach wird der `HostRenderer` erstellt, der das parallele Rendern des Bildes steuert und für jeden Pixel die Funktion `render_pixel` aufruft. Die Funktion `render_pixel` wiederum erzeugt einen Strahl durch die Mitte des entsprechenden Pixels und ruft `trace_recursive` auf, welches der Einstieg in das rekursive Whitted-Style Raytracing ist.

- Die meisten für das Raytracing relevanten Dateien befinden sich in den Unterordnern `cglib/include/cglib/rt` und `cglib/src/rt`. Die Kernfunktionen der Raytracers sind in `renderer.h` deklariert und - bis auf die Funktionen, die Sie in diesem Übungsblatt implementieren sollen - in `renderer.cpp` definiert. So z.B. die Funktion `shoot_ray`, die den nächsten Schnittpunkt eines Strahls mit allen Objekten in der Szene bestimmt.

- In `material.h` finden Sie die gesamten Materialparameter. Wundern Sie sich nicht über die Klasse `ConstTexture`, die eine konstante Textur (also einen RGB-Wert) darstellt. Im nächsten Übungsblatt werden richtige Texturen behandelt. Die Klasse `MaterialSample` ist ein an einer bestimmten Stelle ausgewertetes und potenziell texturiertes Material, das nur noch RGB-Werte oder elementare Datentypen enthält.

- Die Klasse `Intersection` in `intersection.h` kapselt alle wichtigen Informationen eines Schnittes von Strahl und Objekt. Einige dieser Informationen werden erst in späteren Übungsblättern benötigt.

- Sie können die Kamera in dieser Aufgabe je nach Szene mit Maus und Tastatur steuern. Entweder die Kamera bewegt sich durch einen Mausdrag um einen Referenzpunkt (Zoom mit Mausrad) oder die Kamera lässt sich wie in einem First-Person-Shooter steuern.

- In der GUI haben Sie die Möglichkeit mehrere Render-Modi einzustellen. Sie finden z.B. einen Modus, der die Anzahl an Strahlschnitten pro Pixel visualisiert. Außerdem gibt es den Modus `Desaturate`, der das gerenderte Farbbild in ein Graustufenbild umwandelt.

- Der Modus `Desaturate` ist besonders nützlich, wenn Sie in der GUI Stereo Rendering aktivieren. Damit können Sie sich das Bild mithilfe der in der Übung ausgeteilten Brille mit verstärktem Tiefeneindruck ansehen.

- Wenn Sie das Programm mit `--help` auf der Kommandozeile aufrufen, erhalten Sie einen Überblick über die Parameter mit denen das Programm gestartet werden kann. So können Sie z.B. die Auflösung des Bildes reduzieren, falls Ihnen das Rendern eines Bildes zu lange dauert. Wenn Sie die Größe des Fensters verändern, wird das gerenderte Bild entsprechend skaliert.

- **Hinweis:** Sie erhalten einige Referenzbilder, die zeigen, wie Ihre Ausgabebilder nach den einzelnen Aufgabenteilen nach korrekter Implementierung aussehen sollten. Eine Benennung wie `_after_c.png` bedeutet, dass das Bild so aussehen sollte, wenn Sie alle Aufgabenteile bis einschließlich Aufgabenteil c) korrekt implementiert haben.

- **Hinweis:** Der von Ihnen geschriebene Code sollte sich ausschließlich in den dafür vorgesehenen Funktionen in der Datei `exercise_02.cpp` befinden.

a) **(4 Punkte)** In der ersten Aufgabe sollen Sie in der Datei `exercise_02.cpp` in der Funktion `intersect_sphere` einen Strahl-Kugel-Schnitttest implementieren. Die Deklaration der zu implementierenden Funktion und eine Implementierung eines Strahl-Ebene-Schnitttest finden Sie in der Datei `intersection_tests.h` in der `cglib`.

b) **(6 Punkte)** Erweitern Sie nun den Raytracer mit dem Phong-Beleuchtungsmodell. Wir werden das Beleuchtungsmodell ein wenig gegenüber der Vorlesung abwandeln. Das Auswerten der Beleuchtung soll in `evaluate_phong` in der Datei `exercise_02.cpp` implementiert werden. Dabei soll über alle Lichtquellen iteriert und die Beleuchtung, bestehend aus ambientem, diffusem und spekularem Term, in der Variablen `contribution` akkumuliert werden. Für den diffusen und spekularen Term muss zusätzlich die Sichtbarkeit der Lichtquelle zum Oberflächenpunkt geprüft werden. Benutzen Sie dafür die vorhandene Funktion `visible`, die in der Datei `renderer.h` definiert und in der Datei `renderer.cpp` implementiert ist. Deaktiviert man mit der GUI-Checkbox „Shadows" die Sichtbarkeitstests, so soll die gesamte Szene beleuchtet dargestellt werden (siehe auch `box_after_b_no_shadow.tga`. Die Lichtstärke soll zudem mit dem Abstandsquadrat abfallen, um eine realistische Beleuchtung zu realisieren. Das übergebene `MaterialSample` enthält den diffusen (`k_d`), spekularen (`k_s`) und ambienten

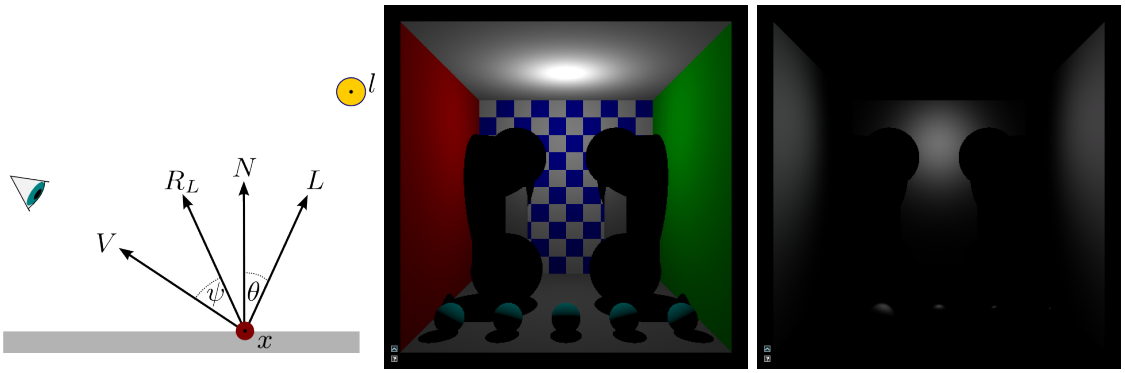


Abbildung 2: Lage der Vektoren für das Phong-Beleuchtungsmodell. Diffuser und spekularer Anteil der Beleuchtung.

(`k_a`) Materialparameter sowie den Phong-Exponent `n`. Die Beleuchtung an einem Punkt x durch eine Lichtquelle l lässt sich berechnen mit

$$\frac{E_l(-L) \cdot S(x,l) \cdot O(\cos\theta)}{\|x-l\|^2} \left(k_d \cdot \max(0, \cos\theta) + k_s \cdot \max(0, \cos\psi)^n\right) + \frac{P_l}{\|x-l\|^2} k_a$$

Dabei ist $\|x-l\|^2$ der quadrierte euklidische Abstand zwischen x und l, $E_l(\omega)$ das emittierte Licht der Lichtquelle in Richtung ω, P_l die Gesamtenergie (Power, Leistung) der Lichtquelle und

$$S(x,l) = \begin{cases} 1, & \text{wenn } l \text{ von } x \text{ aus sichtbar} \\ 0, & \text{sonst} \end{cases}, \text{ und } O(x) = \begin{cases} 1, & \text{wenn } x > 0 \\ 0, & \text{sonst} \end{cases}.$$

Alle Vektoren müssen dabei normiert sein. Sie sind in Abbildung 2 illustriert. Die fünf Kugeln im vorderen Bereich der Szene haben von links nach rechts aufsteigende Phong-Exponenten (die linke Kugel ist komplett diffus). Das spekulare Highlight sollte daher wie in Abbildung 2 immer kleiner werden. In dieser Abbildung sehen Sie einmal die diffuse und einmal die spekulare Komponente der Beleuchtung.

Hinweis: Achten Sie darauf, dass die dafür vorgesehenen Komponenten über die GUI-Parameter ein- und ausgeschaltet werden können.

c) **(2 Punkte)** In dieser Aufgabe sollen Sie ein Spotlight (Scheinwerfer) implementieren. In der Datei `light.h` in der `cglib` finden Sie die abstrakte Klasse `Light` und die schon implementierte Klasse `PointLight`. Ein Spotlight hat zusätzlich zu Position und Leistung P noch eine Richtung d und einen Falloff α, der eine Art Öffnungswinkel für das Spotlight darstellen soll. Implementieren Sie in der Datei `exercise_02.cpp` in der Funktion `SpotLight::getEmission` die Abstrahlcharakteristik eines Spotlights definiert durch

$$E(\omega) = P \cdot (\alpha + 2) \cdot \max(0, \cos\theta)^\alpha, \text{ wobei } \cos\theta = \langle \omega, d \rangle$$

Um den Effekt des Spotlights zu sehen, können Sie in der GUI die Szene wechseln. In Abbildung 3 ist die alternative Szene einmal ohne und einmal mit Spotlight dargestellt. Über die GUI lässt sich die Abschwächung (Falloff) des Spotlights verändern und das Spotlight ein- und ausschalten.

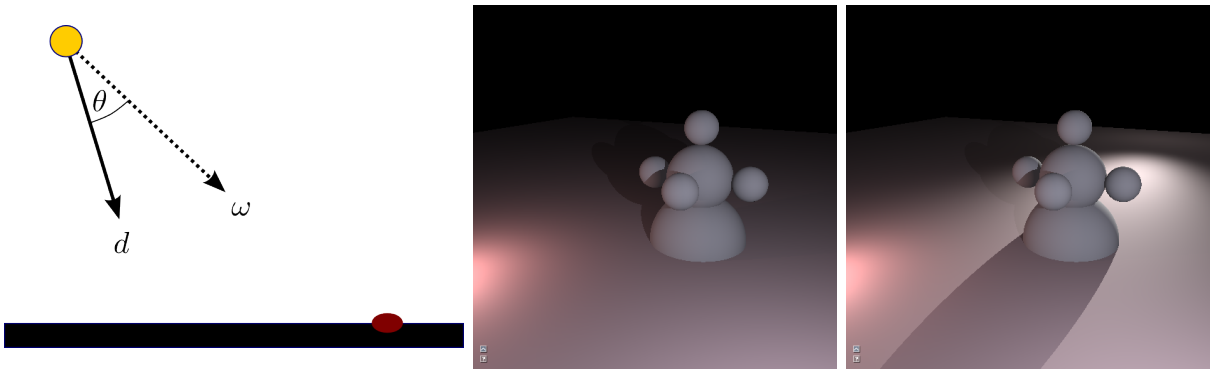


Abbildung 3: Lage der Vektoren für die Beleuchtung mit einem Spotlight. Rendering ohne und mit Spotlight.

d) **(2 Punkte)** In dieser Aufgabe soll der Raytracer um die rekursive Behandlung von perfekter Reflexion erweitert werden. In der Funktion `evaluate_reflection` sollen Sie dazu den Reflexionsstrahl erstellen und rekursiv die Funktion `trace_recursive` aufrufen. Stellen Sie sicher, dass in `trace_recursive` die Rekursion ab der Tiefe `max_depth` abgebrochen wird ohne den Code von `trace_recursive` zu verändern. Zum Erstellen des Reflexionsstrahls können Sie die Funktion `reflect` benutzen, die in `renderer.h` deklariert und in `renderer.cpp` implementiert ist. Wie in Abbildung 4 illustriert, kann durch numerische Ungenauigkeiten Selbstverschattung (Self-Intersection) auftreten. Um diese Artefakte zu vermeiden, sollte der Ursprung des Strahls ein wenig in Richtung der Strahlrichtung R verschoben werden. Der tatsächliche Startpunkt des Strahls ist dann

$$o = p + \epsilon \cdot R \,,$$

wobei R normiert ist. Als Wert für ϵ nehmen Sie bitte `data.context.params.ray_epsilon`, welcher sich über die GUI einstellen lässt.

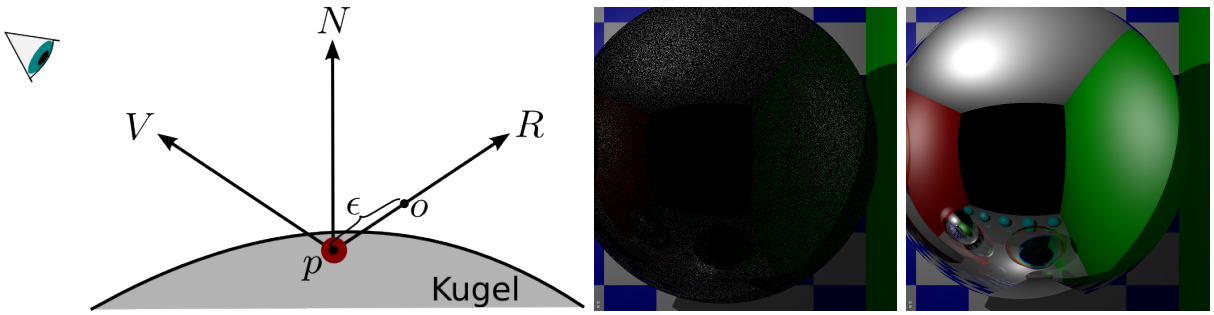


Abbildung 4: Lage der Vektoren für die Reflexion. Selbstverschattung (Self-Intersection) durch numerische Ungenauigkeiten.

e) **(2 Punkte)** Analog zur Reflexion sollen Sie nun in der Funktion `evaluate_transmission` in der Datei `exercise_02.cpp` die rekursive Verfolgung von Transmissionsstrahlen implementieren. Wie stark ein Strahl bei der Transmission abgelenkt wird, hängt vom relativen Brechungsindex η (eta) der angrenzenden Medien ab. Der relative Brechungsindex ist in `MaterialSample` vorhanden und muss von Ihnen nicht verändert werden. Benutzen Sie zum Berechnen der Transmissionsrichtung die Funktion `refract`, welche in der Datei `renderer.h` deklariert und in `renderer.cpp` definiert ist. Die Funktion gibt `false` zurück, falls keine Transmission möglich ist (Stichwort Totalreflexion). Geben Sie in diesem Fall den Wert `(0,0,0)` zurück. Auch bei der Transmission muss Selbstverschattung vermieden werden.

f) **(2 Punkte)** Die Glaskugeln in der Szene sehen noch nicht sehr schön aus. Insbesondere

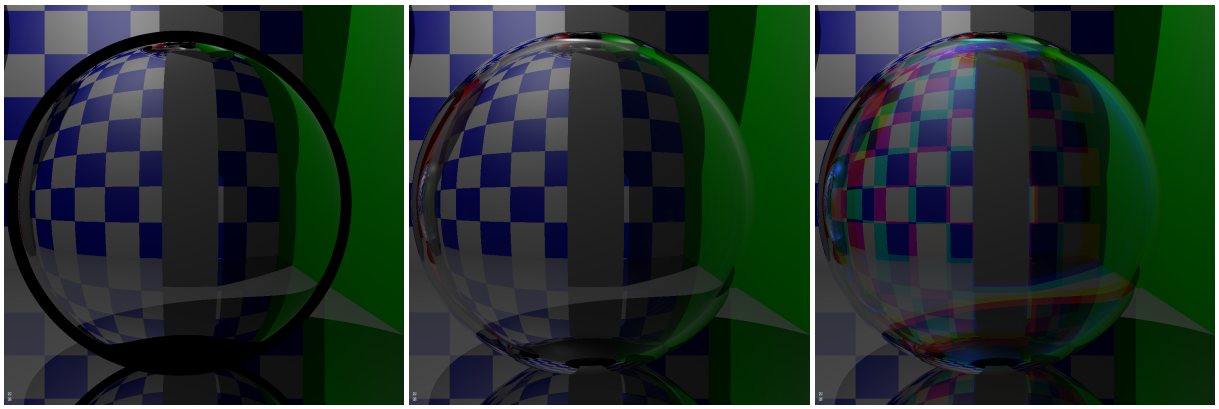


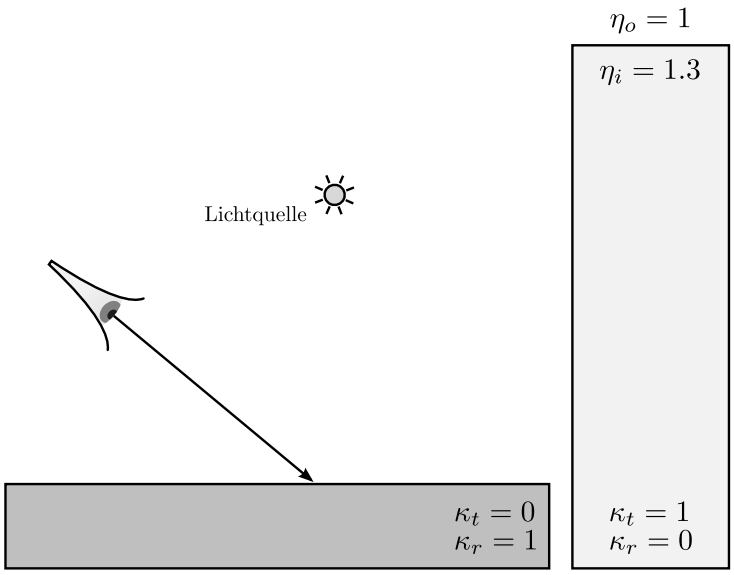
Abbildung 5: Einfluss des Fresnel-Effekts und der Dispersion auf eine Glaskugel.

die schwarzen Ränder (in Abbildung 5 links) sind nicht besonders realistisch. Diese Ränder entstehen, da der Einfallswinkel des Sichtstrahls sehr groß ist und in diesem Winkelbereich kaum noch Transmission stattfindet. Sie sollen dies beheben, indem Sie in der Funktion `handle_transmissive_material_single_ior` abhängig vom Fresnel-Term Transmission und Reflexion kombinieren. Der Fresnel-Term $F \in [0, 1]$ gibt an, wie groß der Anteil der Energie ist, die bei einem bestimmten Einfallswinkel und relativen Brechungsindex reflektiert (F) und transmittiert ($1 - F$) wird. Berechnen Sie den Fresnel-Term mithilfe der Funktion `fresnel`, die in `renderer.h` deklariert und in `renderer.cpp` definiert ist. Erzeugen Sie dann mithilfe von `evaluate_reflection` und `evaluate_transmission` einen Reflexionsstrahl und einen Transmissionstrahl und kombinieren Sie die Ergebnisse additiv gewichtet mit F und $1 - F$. Wenn Sie in der GUI die Berücksichtigung des Fresnel-Effekts aktivieren, sollten Ihre Glaskugeln wie im mittleren Bild in Abbildung 5 aussehen.

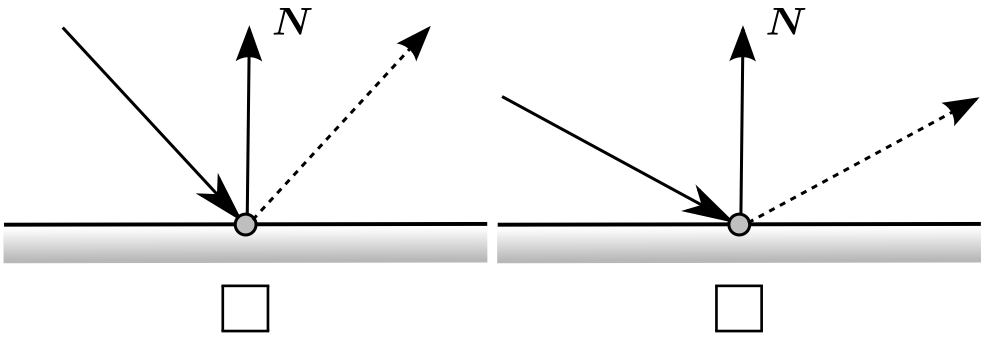
g) **(2 Punkte)** Der relative Brechungsindex in `MaterialSample` ist ein dreidimensionaler Vektor. Dies liegt daran, dass der Brechungsindex üblicherweise abhängig von der Wellenlänge ist. Das führt z.B. bei einem Prisma dazu, dass blaues Licht stärker gebrochen wird als rotes Licht. Diesen Effekt nennt man Dispersion. Um Dispersion korrekt zu behandeln, müsste man spektrales Rendering betreiben, bei dem jeder Strahl eine bestimmte Wellenlänge zugeteilt bekommt. Dies ist aber im Rahmen der Übung zu kompliziert und Sie sollen daher eine vereinfachte Form von Dispersion in der Funktion `handle_transmissive_material` implementieren. Rufen Sie dazu drei mal die Funktion `handle_transmissive_material_single_ior` mit den jeweiligen Komponenten von `eta_of_channel` auf und kombinieren Sie die Ergebnisse. Wenn Sie in der GUI Dispersion aktivieren sollten Ihre Glaskugeln wie im rechten Bild in Abbildung 5 aussehen.

2 Reflexion und Transmission (Theorie, keine Abgabe)

a) Zeichnen Sie in der folgenden Grafik für den eingezeichneten Primärstrahl alle Sekundärstrahlen ein, die beim Whitted-Style-Raytracing erzeugt werden (auch die Strahlen, die keine in der Grafik sichtbaren Objekte treffen). Die dunklere Box soll, wie ein perfekter Spiegel, sämtliches Licht *reflektieren*. Die hellere Box soll, wie perfektes Glas, sämtliches Licht *transmittieren*. Der Brechungsindex der helleren Box ist mit η_i gegeben, die Brechzahl des umgebenden Mediums ist η_0.



b) Die folgenden Skizzen zeigen zwei Lichtstrahlen mit unterschiedlichem Einfallswinkel die an einer *spekularen Glasoberfläche* reflektiert werden (der Vektor N ist die Normale der Oberfläche). Kreuzen Sie den Fall an, in dem ein größerer Teil des einfallenden Lichts reflektiert wird!



c) Wie nennt man das physikalische Gesetz oder Prinzip, welches den Zusammenhang zwischen Einfallswinkel und Reflektivität beschreibt?

d) Wie nennt man das physikalische Gesetz oder Prinzip, welches die Richtungsänderung eines Lichtstrahls beim Übergang in ein anderes Medium beschreibt?

e) Welche Bedingung muss gelten, damit beim Übergang eines Lichtstrahls von einem Medium mit Brechungsindex η_0 in ein Medium mit Brechungsindex η_1 Totalreflexion auftreten kann?

Abgabe

Laden Sie die Datei `exercise_02.cpp` in Ilias hoch.

Framework

Wir werden für jedes Übungsblatt ein Framework bereitstellen, das Sie im Ilias-Kurs unter `https://ilias.studium.kit.edu/goto.php?target=crs_607961&client_id=produktiv` herunterladen können. Das Framework nutzt C++ 11 und wird unter Linux getestet. Es ist allerdings auch unter Windows mit Visual Studio 2013 lauffähig.

Sie können das heruntergeladene Archiv unter Linux mit dem Befehl

```
$ unzip archiv.zip
```

entpacken, wobei Sie `archiv.zip` durch den jeweiligen Dateinamen ersetzen müssen.

Grundsätzlich wird das Framework immer ein Unterverzeichnis `cglib` enthalten. Sie dürfen und sollen den Quellcode in diesem Unterverzeichnis lesen.

Je nach Aufgabe wird es auch ein zweites Unterverzeichnis geben. Für das vorliegende Übungsblatt heißt dieses `02_whitted`. Hier werden Sie Ihre Lösung programmieren.

Die Dateien `VirtualMachine.txt` und `Kompilieren.txt` enthalten Informationen darüber, wie sie die Virtuelle Maschine zur Übung installieren und das Framework kompilieren. Bitte lesen Sie diese Informationen.

Achtung: Abgegebene Lösungen müssen in der VM erfolgreich kompilieren und lauffähig sein, ansonsten vergeben wir 0 Punkte. Insbesondere darf Ihre Lösung nicht abstürzen.

Allgemeine Hinweise zur Übung:

- Scheinkriterien: Sie benötigen 60% der Punkte aus den Praxisaufgaben.
- Die theoretischen Aufgaben bedürfen üblicherweise *keiner* elektronischen Abgabe.
- Die Aufgaben müssen in Ilias bis spätestens Montag, 28.11.2016, 11:00 Uhr abgegeben werden.
- Die Abgabe muss im Ordner `build` mit `cmake ../ && make` in der bereitgestellten VirtualBox VM kompilieren, andernfalls wird die Aufgabe mit 0 Punkten bewertet.
- Da nur einzelne Dateien abgegeben werden, müssen diese kompatibel zu unserer Referenzimplementation bleiben. **Verändern Sie daher wirklich nur die Dateien, die auch abgegeben werden müssen**, bzw. *nicht* die mitgelieferten Funktionsdeklarationen! Sie können allerdings in in den *abzugebenden* Dateien gerne Hilfsfunktionen definieren und benutzen.
- Sie dürfen sehr gerne untereinander die Aufgaben diskutieren, allerdings muss jeder die Aufgaben *selbst* lösen, implementieren und abgeben. Plagiate bewerten wir mit 0 Punkten.
- Wenden Sie sich bei Fragen an einen Übungsleiter. Unsere Büros sind in Gebäude 50.34.

Johannes Meng	Raum 142	meng@kit.edu
Emanuel Schrade	Raum 140	schrade@kit.edu
Florian Simon	Raum 142	florian.simon@kit.edu