Karlsruher Institut für Technologie
Lehrstuhl für Computergrafik

Prof. Dr.-Ing. Carsten Dachsbacher
Dipl.-Inf. Christoph Schied, M.Sc. Emanuel Schrade

Bonusübungsblatt zur Vorlesung Computergraphik im WS 2016/17

Abgabe bis **Freitag, 17.02.2017**, 11:00 Uhr.

Dieses Übungsblatt ist optional. Mit der freiwilligen Abgabe können Sie bis zu 20 Bonuspunkte erreichen. In diesem Übungsblatt sollen Sie Effekte implementieren die mit Whitted Style Raytracing nicht möglich sind. Diese Effekte sind weiche Schatten, indirekte Beleuchtung und Tiefenunschärfe. Für jeden implementierten Effekt erhalten Sie 5 Punkte.

Beispielhafte Ausgaben einer kompletten Implementierung sind in den jeweiligen Abbildungen dargestellt. Ihre Implementierung beschränkt sich wieder auf die Datei `exercise_05.cpp`. Lesen Sie bitte das Übungsblatt sorgfältig und machen Sie sich mit dem Code des Frameworks vertraut.

Für die Abgabe sollen Sie 4 Bilder erzeugen, in denen jeweils einer der 4 Effekte deutlich zu sehen ist. Sie dürfen gerne eigene Szenen erstellen oder vorhandenen Szenen modifizieren (z.B. eine bessere Kameraeinstellung vornehmen)! Sie können Bilder erzeugen indem Sie das Framework im nicht-interaktiven Modus benutzen und die Startparameter in der Datei `raytracing_prameters.h` in `cglib` entsprechend anpassen. Alternativ können Sie auch Screenshots des Fensters erstellen.

Konvertieren Sie bitte die Dateien ins PNG-Format und benennen Sie die Dateien nach dem Effekt, der zu sehen sein sollte, d.h.

- `depthoffield.png`,
- `ambientocclusion.png`,
- `softshadows.png` und
- `indirectillumination.png`

Erstellen Sie zur Abgabe ein Archiv `solution.zip`, das die Bilddateien und `exercise_05.cpp` enthält. Laden Sie dieses Archiv in Ilias hoch.

1 *Tiefenunschärfe* — 5 Punkte

Das bisher eingesetzte Kameramodell der idealisierten Lochkamera ist unrealistisch. In der Realität werden Linsen eingesetzt, um Licht aus einer bestimmten Tiefenebene (Fokus) der Szene auf der Bildebene zu fokussieren. Vor und hinter dieser Tiefenebene nimmt die Fokussierung auf der Bildebene ab, was sich in Unschärfe äußert (Abb. 1). Abbildung 2(a) illustriert die möglichen Pfade, auf denen Licht aus der Szene durch eine Linse auf einen Pixel der Bildebene fällt.

In dieser Aufgabe sollen Sie ein leicht vereinfachtes Linsenmodell mit vergleichbarer Wirkung implementieren, wie es in Abbildung 2(b) dargestellt ist. Die Linse ist hier an den Kameraursprung verschoben, was in Realität unmöglich wäre, die betrachteten Strahlen vor der Linse aber

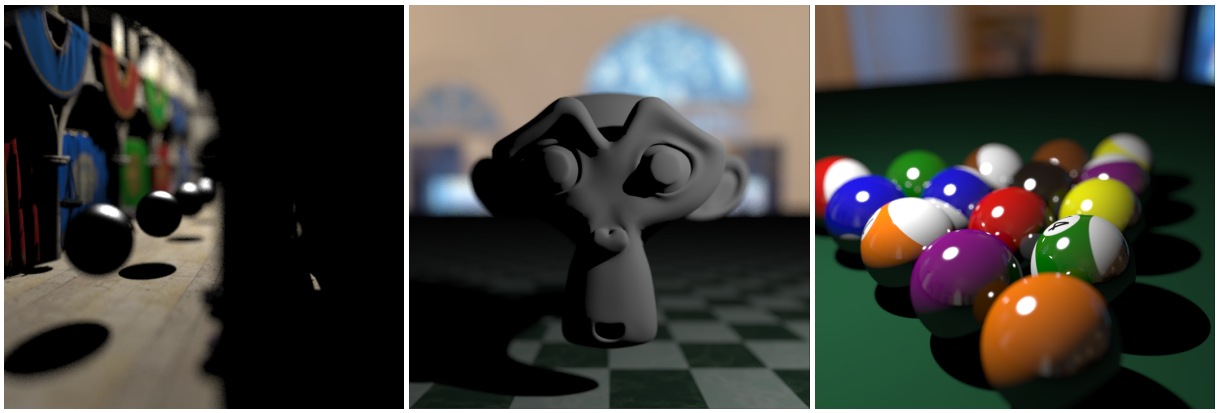

Abbildung 1: Tiefenunschärfe hervorgerufen durch Implementierung eines einfachen Linsenmodells.

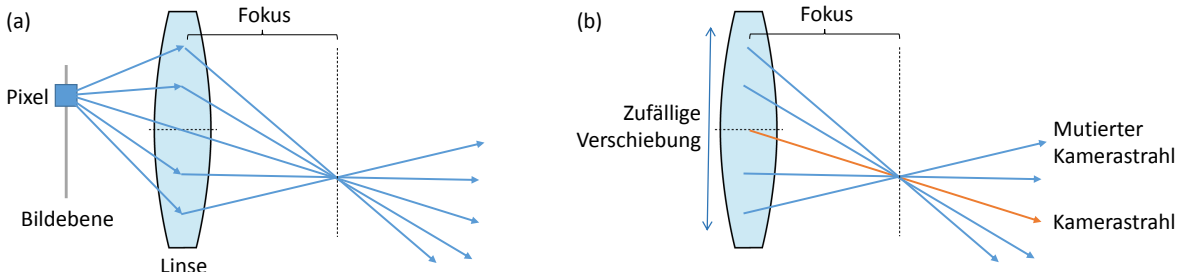

Abbildung 2: (a) Kameramodell mit einer dünnen Linse und (b) zu implementierendes vereinfachtes Kameramodell mit vergleichbarer Wirkung.

nicht verändert. Weiter nehmen wir vereinfachend an, dass alle Strahlen gleichermaßen zum Bild beitragen. Gehen Sie bei der Implementierung in `trace_recursive_with_lens` wie folgt vor:

1. Berechnen Sie zu dem gegebenen Sichtstrahl aus dem Lochkameramodell zunächst den Punkt in der Fokusebene, welcher für alle Sichtstrahlen im Linsenmodell identisch ist. Der Abstand zwischen Fokusebene und Kamera ist durch `data.context.params.focal_length` gegeben. Mit `data.context.scene->camera->get_direction()` können Sie die Kamerarichtung erfragen.

2. Erzeugen Sie `data.context.params.dof_rays` Sichtstrahlen des Linsenmodells, indem Sie den Ursprung des gegebenen Sichtstrahls zufällig auf der kreisförmigen Linse mit Radius `data.context.params.lens_radius` verschieben. Stellen Sie dabei sicher, dass die veränderten Sichtstrahlen weiterhin durch denselben Punkt in der Fokusebene verlaufen. Mit folgender Formel erzeugen Sie gleichverteilte Punkte im Einheitskreis:

$$\begin{aligned} \xi_1,\xi_2 &\in [0,1) \text{ gleichverteilt} \\ r &= \sqrt{\xi_1} \\ \mathbf{p} &= (\ r\ \cos 2\pi\xi_2,\ r\ \sin 2\pi\xi_2\ ) \end{aligned}$$

   Mit der inversen View-Matrix `data.context.scene->camera->get_inverse_view_matrix(data.camera_mode)` lassen sich Punkte aus dem Kameraraum in das Weltkoordinatensystem transformieren. Die Zufallszahlen ξ_1, ξ_2 können Sie mit `data.tld->rand()` ziehen.

3. Tasten Sie die Szene mit jedem der veränderten Sichtstrahlen durch `trace_recursive` ab.

4. Mitteln Sie die Beiträge aller Sichtstrahlen und geben Sie das Resultat zurück.

2 *Ambient Occlusion*	**5 Punkte**

Abbildung 3: Vortäuschung globaler Beleuchtung durch Abschätzung der Verschattung von Umgebungslicht.

Ambient Occlusion ist ein beliebtes Mittel zur Vortäuschung globaler Beleuchtung bei wesentlich reduziertem Rechenaufwand. Die Grundidee ist einfach: An Stelle des aus jeder Richtung einfallenden Lichts wird für jede Richtung die Verschattung durch umliegende Geometrie abgeschätzt. Die Gesamtverschattung aller Richtungen ergibt dann einen Ambient Occlusion-Wert, welcher in etwa die relative Helligkeit des betrachteten Oberflächenpunkts widerspiegelt, wenn aus größerer Distanz aus jeder Richtung gleichmäßiges Licht einfällt.

Im Exkurs der Vorlesung haben Sie die Rendering-Gleichung kennen gelernt. Nimmt man diffus streuende Oberflächen ($f_r(\omega_i, \mathbf{x}, \omega) = \frac{1}{\pi}$) und uniformen Lichteinfall an ($L_i(\mathbf{x}, \omega_i) = V(\mathbf{x}, \omega_i)$, wobei V die Sichtbarkeit der Umgebung von Punkt $\mathbf{x}$ aus in Richtung ω_i angibt), so erhält man den Ambient Occlusion-Koeffizienten k_{ao}:

$$L(\mathbf{x}, \omega) = \int_{\Omega^+} f_r(\omega_i, \mathbf{x}, \omega) L_i(\mathbf{x}, \omega_i) \cos\theta_i d\omega_i \tag{1}$$

$$= \int_{\Omega^+} \frac{1}{\pi} V(\mathbf{x}, \omega_i) \cos\theta_i d\omega_i \tag{2}$$

$$=: k_{ao} \tag{3}$$

Definiert man V als tatsächliche Sichtbarkeitsfunktion, so ergibt sich in geschlossenen Räumen das Problem, dass die Umgebung stets für alle Punkte vollständig verdeckt ist. Um in größeren Räumen den Eindruck von Beleuchtung durch gestreutes Licht zu erwecken, wird V für Ambient Occlusion stattdessen in der Regel mit einer Distanzabschwächung der Verdeckung definiert:

$$V(\mathbf{x}, \omega_i) = \begin{cases} 1, & \text{wenn keine Geometrie in Richtung } \omega_i \\ 1 - \frac{1}{1 + c_{dist} \|\mathbf{x} - \mathbf{x}_n\|^2}, & \text{wenn } \mathbf{x}_n \text{ nächster Verdecker in Richtung } \omega_i \end{cases} \tag{4}$$

Abbildung 4(b) zeigt die Wirkung der Distanzabschwächungsfunktion in V für verschiedene Abschwächungskoeffizienten c_{dist}, Abb. 4(c) zeigt den Verlauf der Abschwächungsfunktion.

Im Exkurs haben Sie gesehen, dass sich Integrationsprobleme mit Hilfe von Monte Carlo-Integration in Abtastprobleme umwandeln lassen, indem das Integral zu einer Bildung eines

Abbildung 4: (a) Geometrische Intuition hinter Ambient Occlusion, (b) Unterschiedlich starke Distanzabschwächung von Verdeckung, (c) übliche Abschwächungsfunktion.

kontinuierlichen Erwartungswertes mit N Stichproben umgedeutet wird:

$$k_{ao} = \int_{\Omega^+} \frac{1}{\pi} V(\mathbf{x}, \omega_i) \cos\theta_i d\omega_i \tag{5}$$

$$\approx \frac{2\pi}{N} \sum_{l=1}^{N} \frac{1}{\pi} V(\mathbf{x}, \omega_i^{(l)}) \cos\theta_i^{(l)} \tag{6}$$

$$= \frac{2}{N} \sum_{l=1}^{N} V(\mathbf{x}, \omega_i^{(l)}) \cos\theta_i^{(l)} \tag{7}$$

Gehen Sie bei der Implementierung in `evaluate_ambient_occlusion` wie folgt vor:

1. Tasten Sie die Verschattung durch umliegende Szenengeometrie mit `data.context.params.ao_rays` Strahlen ab. Generieren Sie hierzu gleichverteilt Richtungen oberhalb des zu beleuchtenden Oberflächenpunktes. Implementieren Sie zuerst in `uniform_sample_sphere` die Erzeugung von gleichverteilten Richtungen auf der Einheitskugel. Dafür können Sie die folgende Formel verwenden:

   $$\begin{aligned} &\xi_1,\xi_2 \in [0,1) \text{ gleichverteilt} \\ &h = 1 - 2\xi_1 \\ &r = \sqrt{1-h^2} \;\; (= \sin\theta, \text{ wobei } \theta = \arccos(h), \text{ vgl.Vl.}) \\ &\mathbf{d} = (\; r\cos 2\pi\xi_2,\; h,\; r\sin 2\pi\xi_2 \;) \end{aligned}$$

   Implementieren Sie nun in `uniform_sample_hemisphere` Rejection Sampling, um die generierten Richtungen auf die Halbkugel oberhalb der Oberfläche zu beschränken (ziehen Sie so lange Richtungen, bis eine Richtung oberhalb der Oberfläche liegt). Zufallszahlen ξ_i können Sie mit `data.tld->rand()` ziehen.

2. Verfolgen Sie mit `max_unobstructed_distance` für jede Richtung einen Verschattungsstrahl in die Szene. Die Funktion gibt Ihnen den Abstand zum nächsten Schnittpunkt auf einem Strahl vom übergebenen Punkt in die übergebene Richtung zurück. Falls kein Treffer gefunden wird, gibt die Funktion den größtmöglichen `float`-Wert zurück.

3. Berechnen Sie die Verschattungssummanden wie in Gleichung (7). Verwenden sie die distanzabgeschwächte Sichtbarkeitsfunktion aus Gleichung (4) mit dem Abschwächungskoeffizienten $c = \texttt{data.context.params.half\_ao\_radius}^{-2}$.

4. Summieren Sie die Verschattungswerte aller Verschattungsstrahlen. Wie im Vorlesungsexkurs zu Monte Carlo-Integration angesprochen und in Gleichung (6) zu sehen, muss das

Ergebnis mit der Ausdehnung des Integrationsbereichs, in diesem Fall der Oberfläche der Einheitshalbkugel, multipliziert werden. Vergessen Sie nicht, das Ergebnis zur Bildung des Erwartungsswertes durch die Anzahl der Strahlen zu teilen.

3 *Flächenlichtquellen und weiche Schatten* **5 Punkte**

Abbildung 5: Weiche Schatten hervorgerufen durch Beleuchtung mit einer Flächenlichtquelle.

In den bisherigen Aufgaben wurden Lichtquellen stets als Punktlichter modelliert. Solche Lichter ohne Ausdehnung verursachen scharfe Schattenkanten, wie sie in der Realität selten anzutreffen sind. In dieser Aufgabe soll deshalb Beleuchtung durch Flächenlichtquellen implementiert werden. Abbildung 5 zeigt, dass eine solche räumliche Ausdehnung der Lichtquelle zu weichen Schatten führt. Abbildung 6 illustriert, wie diese weichen Schatten entstehen.

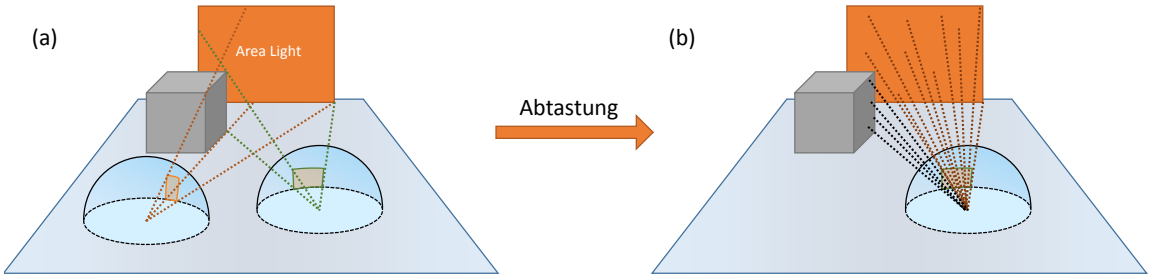

Abbildung 6: (a) Weiche Schatten werden durch eine kontinuierliche Änderung der sichtbaren Fläche ausgedehnter Lichtquellen hervorgerufen. (b) Durch zufällige Abtastung der Lichtquelle kann der Anteil des ankommenden Lichts geschätzt werden.

Für K Flächenlichtquellen mit Flächen A_k lässt sich die Rendering-Gleichung wie folgt umschreiben:

$$L(\mathbf{x},\omega) = \int_{\Omega^+} f_r(\omega_i,\mathbf{x},\omega) L_i(\mathbf{x},\omega_i)\cos\theta_i d\omega_i \tag{8}$$

$$= \sum_{k=1}^{K} \int_{A_k} f_r(-\omega_o,\mathbf{x},\omega) L_e(\mathbf{x}_o,\omega_o) V(\mathbf{x},\mathbf{x}_o) \frac{\cos\theta_i \cos\theta_o}{\|\mathbf{x}-\mathbf{x}_o\|^2} d\mathbf{x}_o \tag{9}$$

In dieser Umformung wurde die Integration über die Lichteinfallsrichtungen umgeschrieben in eine Integration über die Oberflächen der Lichtquellen. V ist die Sichtbarkeitsfunktion, die angibt, ob der Punkt $\mathbf{x}_o$ auf der Flächenlichtquelle von $\mathbf{x}$ aus sichtbar ist. $L_e(\mathbf{x}_o,\omega_o)$ bezeichnet

das an $\mathbf{x}_o$ in Richtung ω_o abgestrahlte Licht, θ_o den Abstrahlwinkel relativ zur Normale der Lichtquelle. $d\mathbf{x}_o$ bezeichnet ein differentielles Flächenstück der Lichtquelle. Die Umschreibung des Integrals ist möglich, weil der differentielle Raumwinkel $d\omega_i$ gerade $\frac{\cos\theta_o}{\|\mathbf{x}-\mathbf{x}_o\|^2}d\mathbf{x}_o$ entspricht.

Die Gleichung lässt sich wieder als Monte Carlo-Problem formulieren:

$$L(\mathbf{x},\omega) = \sum_{k=1}^{K}\int_{A_k} f_r(-\omega_o,\mathbf{x},\omega)L_e(\mathbf{x}_o,\omega_o)V(\mathbf{x},\mathbf{x}_o)\frac{\cos\theta_i\cos\theta_o}{\|\mathbf{x}-\mathbf{x}_o\|^2}d\mathbf{x}_o \tag{10}$$

$$\approx \sum_{k=1}^{K}\frac{\|A_k\|}{N}\sum_{l=1}^{N} f_r(-\omega_o^{(l)},\mathbf{x},\omega)L_e(\mathbf{x}_o^{(l)},\omega_o^{(l)})V(\mathbf{x},\mathbf{x}_o^{(l)})\frac{\cos\theta_i^{(l)}\cos\theta_o^{(l)}}{\|\mathbf{x}-\mathbf{x}_o^{(l)}\|^2} \tag{11}$$

Implementieren Sie in der Funktion `evaluate_illumination` die Beleuchtung durch Flächenlichtquellen wie folgt:

1. Tasten Sie jede der in der Szene vorhandenen Flächenlichtquellen (`data.context.scene->area_lights`) mit `data.context.params.shadow_rays` Strahlen ab, indem Sie gleichverteilt zufällige Punkte auf der Lichtquelle generieren und mit der Funktion `visible` deren Sichtbarkeit vom zu beleuchtenden Oberflächenpunkt aus testen. Die Ausdehnung der Flächenlichtquellen ist durch die Attribute `tangent` und `bitangent` spezifiziert, wobei sich jeder Punkt auf der Lichtquelle schreiben lässt als `position` $+ \xi_1$`tangent` $+ \xi_2$`bitangent` mit gleichverteilten Zufallsvariablen $\xi_1, \xi_2 \in [0, 1)$. Solche Zufallszahlen können Sie mit `data.tld->rand()` ziehen.

2. Berechnen Sie für jede sichtbare Stichprobe den Beitrag des entsprechenden Lichtstrahls. Wie in Gleichung (11) zu sehen, ist im Gegensatz zu Punktlichtquellen bei punktweiser Abtastung von Flächenlichtquellen auch der Abstrahlwinkel an der Lichtquelle entscheidend. Dieser Winkel wird schon in der Methode `getEmission` der Klasse `AreaLight` berücksichtigt. Zur Auswertung des Phong-Beleuchtungsmodells können Sie die Hilfsfunktion `evaluate_phong_BRDF` verwenden, welche Ihnen den Term $f_r(-\omega_o,\mathbf{x},\omega)\cos\theta_i$ zurückgibt.

3. Summieren Sie die Beiträge aller Flächenlichtquellen auf und geben sie den resultierenden Beitrag des beleuchteten Oberflächenpunkts zurück.

4 *Indirekte Beleuchtung* **5 Punkte**

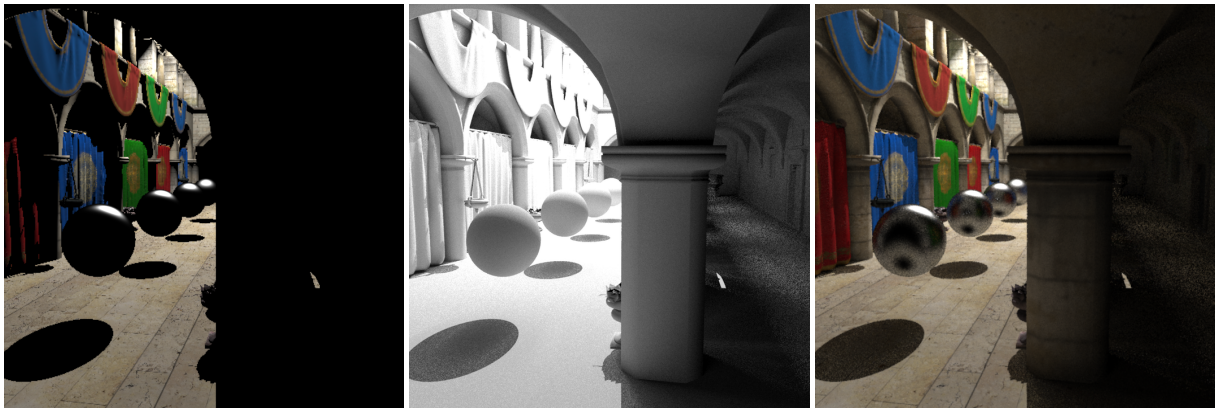

Abbildung 7: Whitted Style Raytracing (links) im Vergleich mit indirekter Beleuchtung und unscharfen Reflexionen.

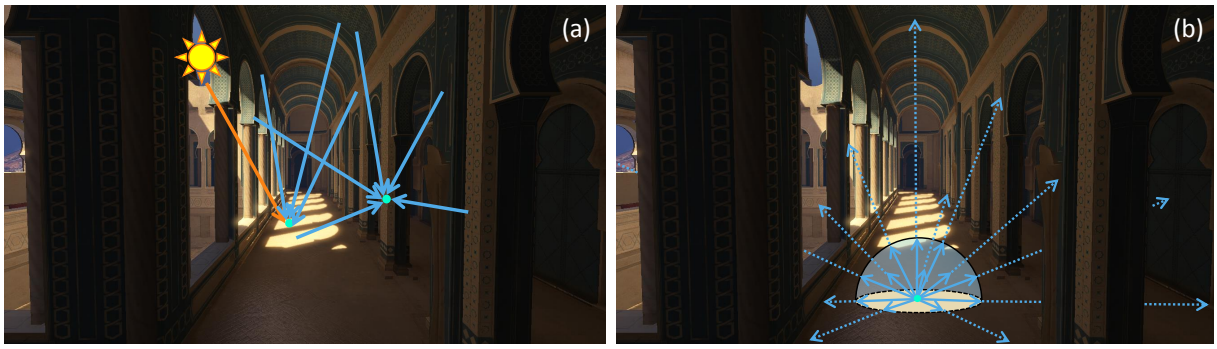

Abbildung 8: (a) Direktes (orange) und indirektes (blau) Licht. (b) Das indirekte Licht an einem Oberflächenpunkt kann durch rekursive Weiterverfolgung von Strahlen in alle möglichen Lichteinfallsrichtungen geschätzt werden.

Bisher wurden Oberflächenpunkte nur durch direktes Licht beleuchtet, welches von den Lichtquellen ausgeht. In der Realität werden Oberflächen zusätzlich auch von indirektem, an anderen Oberflächen gestreutem Licht beeinflusst. Um auch indirektes Licht zu berücksichtigen, muss an jedem Oberflächenpunkt das einfallende Licht für alle möglichen Lichteinfallsrichtungen ausgewertet werden. Die Rendering-Gleichung formalisiert diesen rekursiven Lichttransport:

$$L(\mathbf{x}, \omega) = \int_{\Omega^+} f_r(\omega_i, \mathbf{x}, \omega) L_i(\mathbf{x}, \omega_i) \cos\theta_i d\omega_i \tag{12}$$

$$= \int_{\Omega^+} f_r(\omega_i, \mathbf{x}, \omega) L(\text{raycast}(\mathbf{x}, \omega_i), -\omega_i) \cos\theta_i d\omega_i \tag{13}$$

Da sich beliebig viele solche möglichen Lichteinfallsrichtungen finden lassen, muss das Problem durch Auswahl einer endlichen Anzahl von Stichproben genähert werden:

$$L(\mathbf{x}, \omega) = \int_{\Omega^+} f_r(\omega_i, \mathbf{x}, \omega) L(\text{raycast}(\mathbf{x}, \omega_i), -\omega_i) \cos\theta_i d\omega_i \tag{14}$$

$$= \frac{2\pi}{N} \sum_{l=1}^{N} f_r(\omega_i^{(l)}, \mathbf{x}, \omega) L(\text{raycast}(\mathbf{x}, \omega_i^{(l)}), -\omega_i^{(l)}) \cos\theta_i^{(l)} \tag{15}$$

Gehen Sie bei der Implementierung in `evaluate_illumination` wie folgt vor:

1. Tasten Sie das aus der Szene einfallende Licht mit `data.context.params.indirect_rays` Strahlen ab. Generieren Sie hierzu gleichverteilt Richtungen oberhalb des zu beleuchtenden Oberflächenpunktes. Implementieren Sie zuerst in `uniform_sample_sphere` die Erzeugung von gleichverteilten Richtungen auf der Einheitskugel. Dafür können Sie die folgende Formel verwenden:

$$
\begin{aligned}
&\xi_1,\xi_2 \in [0,1) \text{ gleichverteilt} \\
&\quad h = 1 - 2\xi_1 \\
&\quad r = \sqrt{1-h^2} \;\; (= \sin\theta, \text{ wobei } \theta = \arccos(h), \text{ vgl.Vl.}) \\
&\quad \mathbf{d} = (\; r\,\cos 2\pi\xi_2,\; h,\; r\,\sin 2\pi\xi_2 \;)
\end{aligned}
$$

   Implementieren Sie nun in `uniform_sample_hemisphere` Rejection Sampling, um die generierten Richtungen auf die Halbkugel oberhalb der Oberfläche zu beschränken (ziehen Sie so lange Richtungen, bis eine Richtung oberhalb der Oberfläche liegt). Zufallszahlen ξ_i können Sie mit `data.tld->rand()` ziehen.

2. Verfolgen Sie mit `trace_recursive` für jede Richtung einen Lichteinfallsstrahl in die Szene.

3. Multiplizieren Sie den Beitrag L jedes Lichteinfallsstrahls mit den übrigen Termen in Gleichung (15). Zur Bestimmung des relativen Beitrags eines Lichtstrahls können Sie das Phong-Beleuchtungsmodell mit der Hilfsfunktion `evaluate_phong_BRDF` auswerten, welche den Term $f_r(\omega_i, \mathbf{x}, \omega)\cos\theta_i$ zurückgibt.

4. Summieren Sie die Beiträge aller Lichteinfallsstrahlen. Wie im Vorlesungsexkurs zu Monte Carlo-Integration angesprochen und in Gleichung (15) zu sehen, muss das Ergebnis mit der Ausdehnung des Integrationsbereichs, in diesem Fall der Oberfläche der Einheitshalbkugel, multipliziert werden. Vergessen Sie nicht, das Ergebnis zur Bildung des Erwartungsswertes durch die Anzahl der Strahlen zu teilen.

Abgabe

Laden Sie die Datei `solution.zip` in Ilias hoch. Achten Sie darauf, dass dieses Archiv die von Ihnen bearbeitete Datei `exercise_05.cpp` sowie die Ergenisbilder `depthoffield.png`, `ambientocclusion.png`, `softshadows.png` und `indirectillumination.png` enthält.

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 `05_distributed`. 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 **Freitag, 17.02.2017**, 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.

Christoph Schied	Raum 136	christoph.schied@kit.edu
Emanuel Schrade	Raum 140	schrade@kit.edu