

Bonus-Übungsblatt zur Vorlesung Computergrafik im WS 2021/2022

Abgabe bis Mittwoch, 23.02.2022, 23:59 Uhr

Bonuspunkte

Auf diesem Blatt können Sie Bonuspunkte sammeln. Diese Bonuspunkte können Sie in diesem Semester einsetzen für eine der beiden folgenden Möglichkeiten

- entweder, falls Sie den Schein schon bestanden haben, um einen Bonus für die Klausur (HK oder NK 2022) zu bekommen
 - Der Bonus wird nur auf bestandene Klausuren angerechnet
 - Bei 10 erzielten Bonusblatt-Punkten erhalten Sie garantiert eine Notenstufe Bonus (z.B. 4.0 zu 3.7, 2.7 zu 2.3, 1.3 zu 1.0). Mehr als eine Notenstufe Bonus ist nicht möglich.
 - Bei weniger als 10 erzielten Bonusblatt-Punkten werden die Bonusblatt-Punkte anteilig auf Klausurpunkte umgerechnet.
- oder, falls Sie den Schein knapp nicht bestanden haben, um Bonuspunkte für den Übungsschein zu bekommen. In dem Fall werden Bonusblatt-Punkte 1:1 zu Übungsblattpunkten umgerechnet. Die Bonusblatt-Punkte können auf beide Übungsblatt-Blöcke aufgeteilt werden.

Abgabe

Auf diesem Aufgabenblatt können Sie vorgefertigte Aufgabenvorschläge (siehe unten) bearbeiten, und/oder sich selbst Features aussuchen, um die Sie einen Distributed Raytracer erweitern.

Da auf diesem Blatt viele Auswahlmöglichkeiten bestehen, haben wir im ILIAS-Kurs *zwei Bonusaufgaben* erstellt, damit wir die Abgaben besser automatisiert trennen können. Bitte laden Sie Ihre Lösungen für Distributed Raytracing-Aufgaben (Aufgaben 1-5) in der Bonusaufgabe "Bonus: Distributed Raytracing" hoch, und Ihre Lösungen für Aufgaben 6-9 in "Bonus: Sonstige" hoch. Falsch zugeordnete Abgaben werden nicht bewertet. Genaueres zur Abgabe finden Sie auf Seite 2 und 12.

Framework Die vorgeschlagenen Aufgaben basieren teilweise auf verschiedenen vergangenen Versionen des Frameworks, diese können Sie von den entsprechenden Übungsaufgaben im ILIAS herunterladen. Für eigene Features (Aufgabe 1) sowie die vorgeschlagenen Aufgaben zum Raytracing (Aufgabe 2-5) stellen wir mit diesem Blatt gemeinsam ein extra Framework bereit.

ACHTUNG Abgegebene Lösungen müssen in der VM erfolgreich kompilieren und lauffähig sein, ansonsten vergeben wir 0 Punkte! Insbesondere darf Ihre Lösung nicht abstürzen. Bitte testen Sie vorher ob Ihre Lösung in der VM kompiliert! Die Abgabe muss im Ordner `build` mit `cmake .. / && make` in der VM kompilieren.

Sie dürfen sehr gerne untereinander die Aufgaben diskutieren, allerdings muss jede*r die Aufgaben *selbst* lösen, implementieren und abgeben. Plagiate bewerten wir mit 0 Punkten.

Kontakt: Übungsleitung `cg-uebung-ws2122@kit.edu`

Abgabemodus für Aufgaben 1-5

Laden Sie Ihre Lösungen für die folgenden Aufgaben für die ILIAS-Bonusaufgabe "Bonus: Distributed Raytracing" hoch! Abgaben, die in "Bonus: Sonstige" hochgeladen werden, werden nicht berücksichtigt.

Laden Sie dafür *eine einzige*¹ zip-Datei **solutions.zip** hoch. Die zip-Datei soll folgendes enthalten:

- eine Text-Datei namens **readme.txt**, in der Sie auflisten, welche Aufgaben Sie bearbeitet haben
- falls Sie eine oder mehrere der Aufgaben 2-5 bearbeitet haben: *einen (einzigen)* Ordner **vorschlaege**. Dieser Ordner soll enthalten:
 - die Datei **exercise_05.cpp**
 - Je bearbeiteter Aufgabe eine Datei **ansatz_X.txt** X=2,3,4,5, in der Sie in einigen Sätzen beschreiben, wie Sie die Aufgabe gelöst haben.
 - Je bearbeiteter Aufgabe ein Bild **effekt_X.png**, in dem man den Effekt beobachten kann.
- falls Sie einen oder mehrere eigene Effekte implementiert haben: *Je implementiertem Effekt* einen Ordner **effektnname**. Dieser Ordner soll enthalten:
 - das komplette Framework (alle Quelldateien des Programms) inklusive neu erstellter Dateien, sodass das abgegebene Framework ohne Anpassung direkt in der VM kompiliert und ausgeführt werden kann.
 - eine Datei **aenderungen_effektnname.txt**, in der Sie auflisten, welche Dateien Sie verändert haben, und welche Dateien Sie neu erstellt haben
 - eine Datei **ansatz_effektnname.txt**, in der Sie in einigen Sätzen beschreiben, wie Sie die Aufgabe gelöst haben.
 - *ein* Bild **effektnname.png**, in dem man den Effekt beobachten kann.

Bitte beachten Sie: Wir bewerten die Aufgaben unabhängig voneinander. Es darf *keine* Abhängigkeiten zwischen den Ordnern geben!

¹Sie dürfen natürlich Ihre Abgabe aktualisieren, d.h. als Sie mehrere Dateien names **solutions.zip** hochladen, werten wir die späteste Abgabe

1 Freie Effekte im Distributed Raytracer

Variabel viele Punkte

1+2 naivem Whitted Style Raytracing 18%

Hier können Sie Effekte implementieren, die mit naivem Whitted Style Raytracing nicht möglich sind. Die Beurteilung richtet sich nach Aufwand und Qualität der Effekte. In Aufgaben 2-5 finden Sie beispielhaft 4 Effekte mit möglichen Implementierungsansätzen und beispielhaften Ergebnissen (Tiefunschärfe, Ambient Occlusion, weiche Schatten und indirekte Beleuchtung). Sie dürfen auch andere Effekte wählen (zur Inspiration siehe z.B. die Liste weiter unten). Falls Sie sich nicht sicher sind, wie umfangreich Sie Ihre(n) gewählte(n) Effekt(e) implementieren sollen, welche Punktzahl Sie jeweils maximal erwarten können, oder ob ein hier nicht aufgelisteter Effekt ebenfalls geeignet wäre, wenden Sie sich an die Übungsleitung: `cg-uebung-ws2122@kit.edu`, damit wir vorab klären können, welcher Leistungs- und Bonuspunkte-Umfang dafür erwartet werden kann, und damit wir Ihnen Rückmeldung zum Schwierigkeitsgrad und ggf. Hinweise zu Umfang und zur Lösung geben können.

Als Grundlage können Sie das Framework für dieses Bonusblatt verwenden.

Für die Effekte aus Aufgaben 2-5 soll sich die Implementierung auf die Datei `exercise_05.cpp` beschränken. Für andere Effekte dürfen Sie auch andere Dateien des Frameworks verändern und neue Dateien hinzufügen.

Für die Abgabe sollen Sie je Effekt ein Bild erzeugen, in welchem der Effekt 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_parameters.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 wie in den Abgabemodalitäten beschrieben, also `effektnname.png` für eigene Effekte und `effekt_X.png` mit X=2,3,4,5 für Aufgaben 2-5.

Hier sind einige Vorschläge für weitere interessante Effekte:

- Eine interessantere BRDF einbauen als die Phong-BRDF
- Einen Path Tracer schreiben anstatt eines Distributed Ray Tracers
- Rauschentfernung auf dem finalen Bild, z.B. durch einen Bilateralen Filter angewandt auf die Ausgabe des Distributed Raytracers
- Kantenerhaltender Filter
- Das Thinelens-Kameramodell
- Displacement Mapping
- Image based Lighting (IBL)
- alternative Beschleunigungsstrukturen (Octree, ...)

2 Raytracing Beispiel: Tiefunschärfe

Bis zu 3 Punkte



Abbildung 1: Tiefunschärfe hervorgerufen durch Implementierung eines einfachen Linsenmodells.

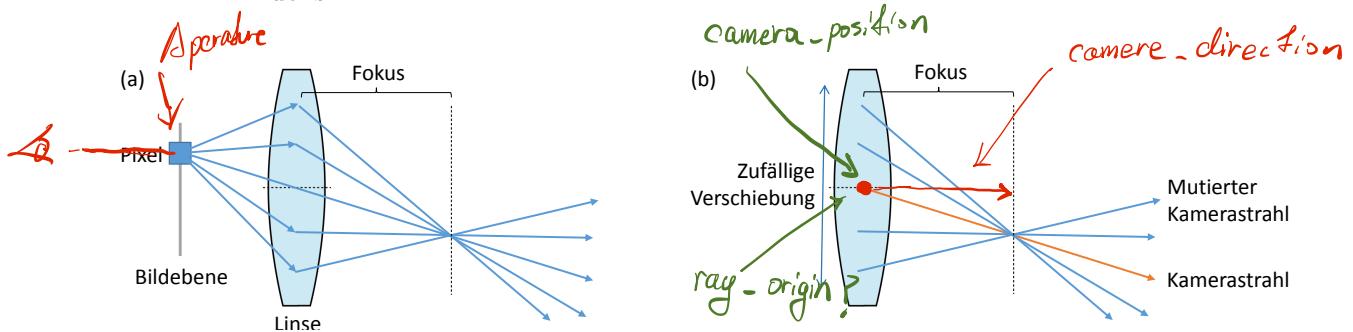


Abbildung 2: (a) Kameramodell mit einer dünnen Linse und (b) zu implementierendes vereinfachtes Kameramodell mit vergleichbarer Wirkung.

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 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.get_active_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.get_active_scene()->camera->get_inverse_view_matrix(data.camera_mode)` lassen sich Punkte aus dem Kamera-Raum 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.

3 Raytracing Beispiel: Ambient Occlusion

Bis zu 2 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 einfalldenden 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} :

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

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

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:

$$V(\mathbf{x}, \omega_i) = \begin{cases} 1, & \text{wenn keine Geometrie in Richtung } \omega_i \\ 1 - \frac{1}{1+c_{dist}\|\mathbf{x}-\mathbf{x}_n\|^2}, & \text{wenn } \mathbf{x}_n \text{ nächster Verdecker in Richtung } \omega_i \end{cases} \quad (3)$$

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 kontinuierlichen Erwartungswertes mit N Stichproben umgedeutet wird:



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

$$k_{ao} = \int_{\Omega^+} \frac{1}{\pi} V(\mathbf{x}, \omega_i) \cos \theta_i d\omega_i \quad (4)$$

$$\approx \frac{2\pi}{N} \sum_{l=1}^N \frac{1}{\pi} V(\mathbf{x}, \omega_i^{(l)}) \cos \theta_i^{(l)} \quad (5)$$

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

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, z.B. so:

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

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 (6). Verwenden sie die distanzabgeschwächte Sichtbarkeitsfunktion aus Gleichung (3) mit dem Abschwächungskoeffizienten $c = \text{data.context.params.half_ao_radius}^{-2}$.
4. Summieren Sie die Verschattungswerte aller Verschattungsstrahlen. Wie im Vorlesungsexkurs zu Monte Carlo-Integration angesprochen und in Gleichung (5) 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 Erwartungswertes durch die Anzahl der Strahlen zu teilen.



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

In den bisherigen ~~Aufgaben~~, ~~wurde~~ 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.

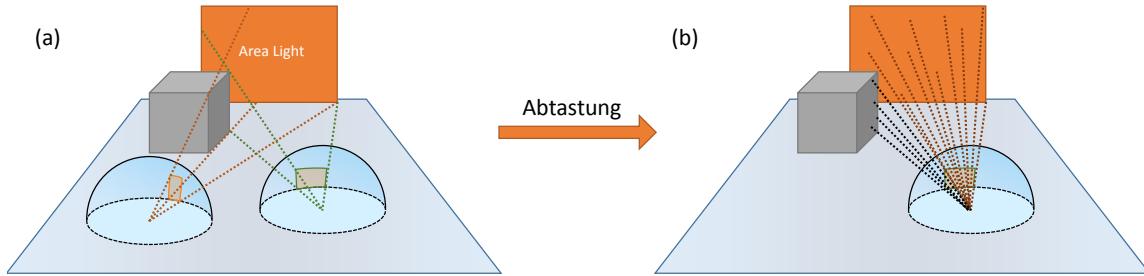


Abbildung 6: (a) Weiche Schatten werden durch eine kontinuierliche Änderung der sichtbaren Fläche ausgedehnter Lichtquellen hervorgerufen. (b) Durch zufällige Abtastung der Lichtquelle kann der Anteil des ankommenden Lichts geschätzt werden.

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

$$L(\mathbf{x}, \omega) = \int_{\Omega^+} f_r(\omega_i, \mathbf{x}, \omega) L_i(\mathbf{x}, \omega_i) \cos \theta_i d\omega_i \quad (7)$$

$$= \sum_{k=1}^K \int_{A_k} f_r(-\omega_o, \mathbf{x}, \omega) L_e(\mathbf{x}_o, \omega_o) V(\mathbf{x}, \mathbf{x}_o) \frac{\cos \theta_i \cos \theta_o}{\|\mathbf{x} - \mathbf{x}_o\|^2} d\mathbf{x}_o \quad (8)$$

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 $\omega_o = \frac{\mathbf{x} - \mathbf{x}_o}{\|\mathbf{x} - \mathbf{x}_o\|}$ 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:

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

$$\approx \sum_{k=1}^K \frac{\|A_k\|}{N} \sum_{l=1}^N \underbrace{f_r(-\omega_o^{(l)}, \mathbf{x}, \omega)}_{\text{Term}} L_e(\mathbf{x}_o^{(l)}, \omega_o^{(l)}) V(\mathbf{x}, \mathbf{x}_o^{(l)}) \frac{\cos \theta_i^{(l)} \cos \theta_o^{(l)}}{\|\mathbf{x} - \mathbf{x}_o^{(l)}\|^2} \quad (10)$$

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.get_active_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 + ξ₁tangent + ξ₂bitangent` 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 (10) 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.

5 Raytracing Beispiel: Indirekte Beleuchtung

Bis zu 4 Punkte

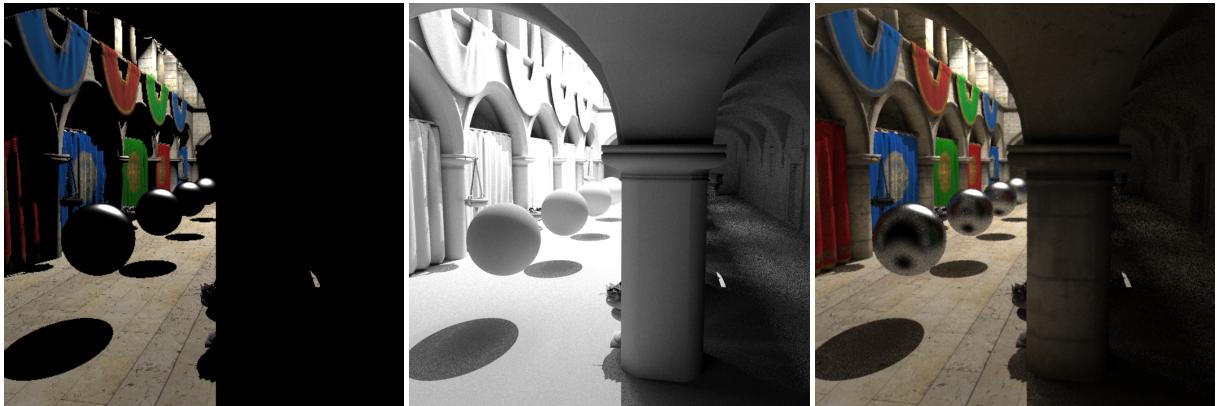


Abbildung 7: Whitted Style Raytracing (links) im Vergleich mit indirekter Beleuchtung und unscharfen Reflexionen.

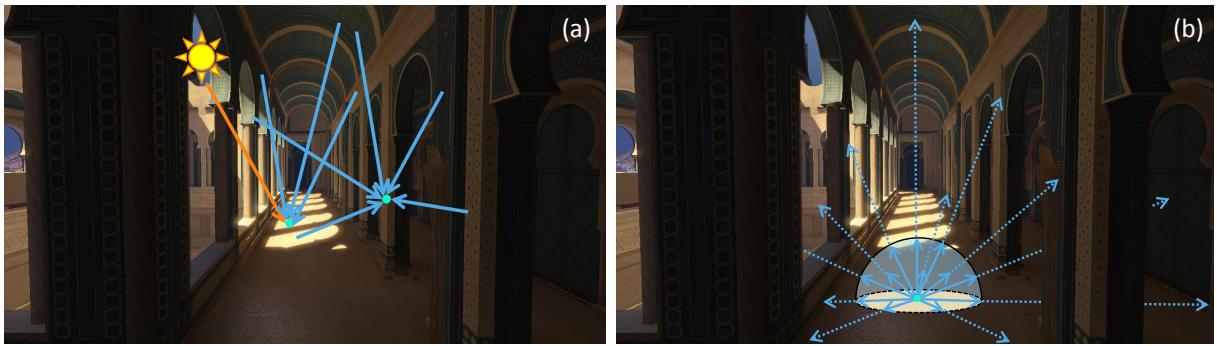


Abbildung 8: (a) Direktes (orange) und indirektes (blau) Licht. (b) Das indirekte Licht an einem Oberflächenpunkt kann durch rekursive Weiterverfolgung von Strahlen in alle möglichen Lichteinfallsrichtungen geschätzt werden.

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:

$$L(\mathbf{x}, \omega) = \int_{\Omega^+} f_r(\omega_i, \mathbf{x}, \omega) L_i(\mathbf{x}, \omega_i) \cos \theta_i d\omega_i \quad (11)$$

$$= \int_{\Omega^+} f_r(\omega_i, \mathbf{x}, \omega) L(\text{raycast}(\mathbf{x}, \omega_i), -\omega_i) \cos \theta_i d\omega_i \quad (12)$$

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

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

$$= \frac{2\pi}{N} \sum_{l=1}^N f_r(\omega_i^{(l)}, \mathbf{x}, \omega) L(\text{raycast}(\mathbf{x}, \omega_i^{(l)}), -\omega_i^{(l)}) \cos \theta_i^{(l)} \quad (14)$$

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:

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

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 `trace_recursive` für jede Richtung einen Lichteinfallsstrahl in die Szene.
3. Multiplizieren Sie den Beitrag L jedes Lichteinfallsstrahls mit den übrigen Termen in Gleichung (14). 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 (14) 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 Erwartungswertes durch die Anzahl der Strahlen zu teilen.

Bitte beachten Sie: Aufgaben 6-9 sind thematisch anders angesiedelt als Raytracing und eignen sich teils zur Bearbeitung zu einem späteren Zeitpunkt (z.B. nach dem OpenGL-Kapitel).

Abgabemodus für Aufgaben 6-9

Laden Sie Ihre Lösungen für die folgenden Aufgaben für die ILIAS-Bonusaufgabe "Bonus: Sonstige" hoch! Abgaben, die in "Bonus: Distributed Raytracing" hochgeladen werden, werden nicht berücksichtigt.

Laden Sie dafür *eine einzige*² zip-Datei **solutions.zip** hoch. Die zip-Datei soll folgendes enthalten:

- eine Text-Datei namens **readme.txt**, in der Sie auflisten, welche Aufgaben Sie bearbeitet haben
- je bearbeiteter Aufgabe einen Ordner **aufgabe_X** mit X=6,7,8,9, der enthält
 - eine Datei **ansatz_X.txt**, in der Sie in einigen Sätzen beschreiben, wie Sie die Aufgabe gelöst haben.
 - die in der jeweiligen Aufgabenbeschreibung genannten Dateien

²Sie dürfen natürlich Ihre Abgabe aktualisieren, d.h. als Sie mehrere Dateien names **solutions.zip** hochladen, werten wir die späteste Abgabe

Als Grundlage für diese Aufgabe können Sie das Framework von Blatt 3 (Texturen) verwenden. Bearbeiten Sie ausschließlich die Datei `exercise_03.cpp`. Für die Abgabe erstellen Sie einen Ordner `aufgabe_6` mit der Datei `exercise_03.cpp` und der Datei `ansatz_6.txt`, in der Sie in wenigen Sätzen beschreiben, wie Sie die Aufgabe gelöst haben.

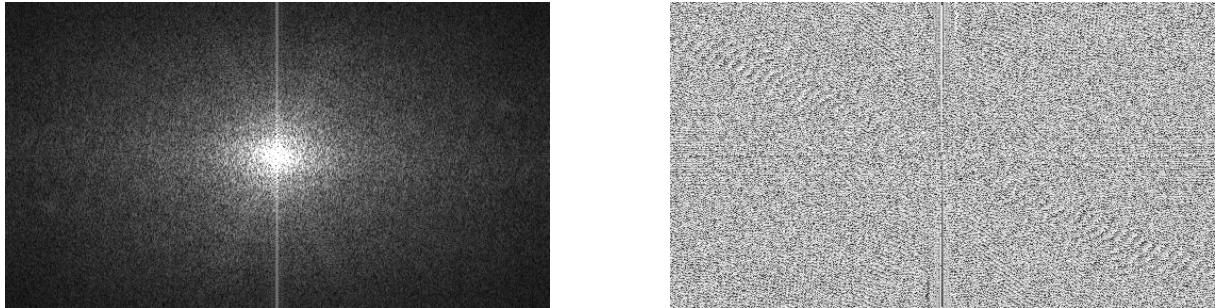


Abbildung 9: Amplitude (links) und Phase (rechts) des Fourierspektrums eines unbekannten Graustufenbildes.

In dieser Bonusaufgabe sollen Sie ein geheimes Graufstufenbild $\{x_{kl}\}$, $(k = 0, \dots, M - 1, l = 0, \dots, N - 1)$ aus einem Fourierspektrum rekonstruieren. Das Fourierspektrum ist in der Datei `assets/mystery.pfm` als Portable Float Map (pfm) gegeben. Dies ist ein primitives Bildformat, das aber nur wenige Image Viewer anzeigen können. Der Real- bzw. Imaginärteil der Fourierkoeffizienten befindet sich in der R- bzw G-Komponente des Bildes.

Die Fourierkoeffizienten $\{\hat{x}_{kl}\}$, $(k = 0, \dots, M - 1, l = 0, \dots, N - 1)$ wurden über die Fouriertransformation

$$\hat{x}_{kl} = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x_{mn} e^{-2\pi im(\frac{k}{M} - \frac{1}{2})} e^{-2\pi in(\frac{l}{N} - \frac{1}{2})} \quad (15)$$

berechnet. Die inverse Transformation ist

$$x_{kl} = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \hat{x}_{mn} e^{2\pi ik(\frac{m}{M} - \frac{1}{2})} e^{2\pi il(\frac{n}{N} - \frac{1}{2})} \quad (16)$$

Das Fourierspektrum und das ursprüngliche Bild haben dieselbe Auflösung. Über die Funktionen `complex_to_image` und `image_to_complex` der Klasse `Image` können Fourierspektren und Bilder ineinander umgewandelt werden. In der Funktion `fourier` in der Datei `main.cpp` wird das Fourierspektrum geladen und die inverse Fouriertransformation aufgerufen. Implementieren Sie diese in der Funktion `DiscreteFourier2D::reconstruct` in `exercise_03.cpp`. Um die Rekonstruktion müssen Sie in der GUI die Szene *Fourier* auswählen. Wählen Sie unter *Image* die gewünschte Variante. Beachten Sie, dass die Bilder nur angezeigt werden können, wenn die bilineare Interpolation (Aufgabe 1c auf Blatt 3) bereits implementiert ist.

Tipp: Die Fouriertransformation ist separierbar, d.h. eine zweidimensionale Transformation lässt sich realisieren, indem man das Bild zuerst horizontal und danach vertikal (oder umgekehrt) eindimensional transformiert. Nutzen Sie dies aus, um die Rekonstruktion zu beschleunigen! Besonders in der VM können die Rechenzeiten sehr lang ausfallen.

Hinweis: Bitte spoilen Sie die Aufgabe nicht für ihre Kommilitonen, indem Sie das rekonstruierte Bild im Forum o.ä. veröffentlichen.

7 Surface Area Heuristic**3 Punkte**

Als Grundlage für diese Aufgabe können Sie das Framework von Blatt 4 (Beschleunigungsstrukturen) verwenden. Bearbeiten Sie für diese Aufgabe die Datei `exercise_04.cpp`. Für die Abgabe erstellen Sie einen Ordner `aufgabe_7` mit der Datei `exercise_04.cpp`, der Datei `ansatz_6.txt`, in der Sie in wenigen Sätzen beschreiben, wie Sie die Aufgabe gelöst haben, und außerdem die weiter unten verlangten Ausgaben `bvh-intersection-time.png` und `aabb-intersection-count.png`.

In dieser Bonusaufgabe sollen Sie die BVH-Konstruktion so umgestalten, dass sie nach der Surface Area Heuristic (SAH) erfolgt. Passen Sie dazu in `exercise_04.cpp` die Methode `build_bvh` nach diesem Verfahren an. Verwenden Sie die Konstruktion mit inkrementeller Berechnung, indem Sie für jede der drei Achsen X , Y , und Z die möglichen Unterteilungen testen. Überprüfen Sie Ihre Implementierung mithilfe der Visualisierungsmodi. Achtung: Da SAH die Komplexität der Konstruktion erhöht, kann insbesondere die Berechnungszeit für die Sponza-Szene sehr lange dauern.

Erstellen Sie beispielhaft Visualisierungen der „BVH Intersection Time“ und der „AABB Intersection Count“ für die Monkey- und/oder Sponza-Szene, und benennen diese `bvh-intersection-time.png` und `aabb-intersection-count.png`. Fügen Sie diese beiden Bilder dem Ordner `aufgabe_7` hinzu.

Denken Sie außerdem an die Datei `ansatz_7.txt`, in der Sie in wenigen Sätzen beschreiben, wie Sie die Aufgabe gelöst haben.

Als Grundlage für diese Aufgabe können Sie das Framework von Blatt 5 (Shader) verwenden. Bearbeiten Sie ausschließlich die Datei `exercise_06.cpp`. Für die Abgabe erstellen Sie einen Ordner `aufgabe_8` mit der Datei `exercise_06.cpp`, der Datei `ansatz_8.txt`, in der Sie in wenigen Sätzen beschreiben, wie Sie die Aufgabe gelöst haben, und außerdem je Teilaufgabe ein Rendering mit fünf Kugeln und fünf verschiedenen Materialeigenschaften, und nennen Sie die Bilder `teilaufgabe-X.png` mit X=1,2,3.

Mit vorgefilterten Environment-Maps lassen sich viele Materialien mit interessanten Reflexions- und Brechungseigenschaften approximativ auch ohne Raytracing und in Echtzeit darstellen. Im Folgenden geht es um die Darstellung von Metallen, sowie von reflektierenden und transmittierenden Dielektrika (z.B. Milchglas). Die willkürlich gewählten Materialkonstanten k_d und k_s in den vorherigen Aufgaben werden dabei durch physikalisch plausible Gesetzmäßigkeiten ersetzt.

Rendern Sie für die folgenden Teilaufgaben jeweils 5 Kugeln nebeneinander mit verschiedenen Material-Eigenschaften (Rauheit, Brechungsindex, etc.). Zur Implementierung der neuen Szene parallel zu Aufgaben im CG-Framework können Sie sich an `MonkeyRender` und `SphereFlakeRender` in `renderers.cpp` orientieren. Vergessen Sie nicht, Ihre neuen Szene in der Main-Funktion in `main.cpp` zu registrieren.

1. Zur Darstellung von Metallen benötigen Sie nur den spekularen Anteil der Umgebungsbeleuchtung. Die Rauheit können Sie nach Belieben über den Phong-Exponenten wählen. Die Reflektivität ist bestimmt durch die (komplexwertigen!) Brechungsindizes des jeweiligen Metalls und die Fresnel-Gleichungen: https://en.wikipedia.org/wiki/Fresnel_equations. Rendern Sie beispielhaft 5 Metalle (Brechungsindizes finden Sie z.B. auf <https://refractiveindex.info/>).

Hinweis: Damit die Metalle ihre typische Farbe erhalten, müssen Sie wellenlängenabhängige Reflektivitäten ausrechnen. Wählen Sie hierzu drei representative Wellenlängen für die Komponenten R,G,B, und werten Sie die Fresnel-Gleichungen für jede Farbkomponente getrennt aus. Sie können außerdem unpolarisiertes Licht annehmen, indem Sie den Durchschnitt der polarisierten Reflektivitäten bilden.

2. Zur Darstellung von Dielektrika (Nicht-Metallen) nehmen Sie nun die diffuse Beleuchtungskomponente hinzu. Die diffuse Komponente streut das Licht, welches nicht gemäß Fresnel-Gleichungen reflektiert wurde (entsprechend ergibt sich $k_d = 1 - k_s$). Auch für Dielektrika finden Sie in der Materialdatenbank (reellwertige!) Brechungsindizes. Für diese Aufgabe können Sie beispielhaft fünf Werte zwischen 1.2 und 1.6 wählen.
3. Zur Darstellung von rauen transmittierenden Materialien wie Milchglas müssen Sie neben der Reflexions- auch eine Transmissionsrichtung ausrechnen (Code dazu hatten Sie bereits in früheren Übungsblättern). Ersetzen Sie in diesem Fall die diffuse Komponente durch eine Transmissionskomponente, welche das vorgefilterte Umgebungslicht der spekularen Komponente auch in die Transmissionsrichtung auswertet. Rendern Sie fünf Kugeln mit verschiedenen Phong-Exponenten!

Hinweis: Um die korrekte Transmissionsrichtung nach Austritt des Transmissionsstrahl aus der Kugel zu erhalten, müssen Sie auch die Transmission an der Kugelrückseite beachten. Vor Auswertung der Umgebungsbeleuchtung wird das Licht also zweimal gebrochen, bei Kugeleintritt und Kugelaustritt. Die entsprechende Oberflächennormale der Kugelrückseite können Sie im Shader durch einen analytischen Kugelschnitttest mit Position und Transmissionsrichtung des Kugeleintritts bestimmen.

9 Partikel mittels Geometry Shader**3 Punkte**

Als Grundlage für diese Aufgabe können Sie das Framework von Blatt 6 (Shadow Mapping) verwenden. Bearbeiten Sie ausschließlich die Datei `particle.geom`. Für die Abgabe erstellen Sie einen Ordner `aufgabe_9` mit der Datei `particle.geom`, der Datei `ansatz_9.txt`, in der Sie in wenigen Sätzen beschreiben, wie Sie die Aufgabe gelöst haben, und einem Screenshot der Raketenpartikel in einem Bild `aufgabe_9.png`.

In dieser Aufgabe sollen Sie dafür sorgen, dass die Partikel der startenden Rakete korrekt gezeichnet werden. Wie Sie in `cglib/src/gl/renderer.cpp` in der Methode `ProceduralLandscapeRenderer::drawParticles` sehen können, werden die Partikel als einfaches vorsortiertes Punkt-Array an OpenGL übergeben. Der Vertex Shader `particle.vert` zerlegt jeden eingehenden Punkt in dessen Weltkoordinate und Radius und gibt beide unverändert weiter. Der Geometry Shader `particle.geom` erhält beide als Eingabe und spezifiziert bereits mit `layout(triangle_strip, max_vertices = 4) out;` einen Triangle Strip von 4 Vertices als Ausgabe.

Ihre Aufgabe ist es, aus den eingehenden Punkten Kamera-ausgerichtete Quadrate zu berechnen, auf welche der vorgegebene Fragment Shader `fire.frag` dann Partikel-Texturen für Feuer und Rauch zeichnen kann. Berechnen Sie hierzu die Normale des kameraausgerichteten Quadrats als Richtung von der gegebenen Partikelposition `world_position[0]`³ zur gegebenen Kameraposition `cam_world_pos`. Die Hochachse des Quadrats erhalten Sie als Kreuzprodukt der Normalen und der mit dem gegebenen Zufallswinkel α (`randomAngle`) gedrehten Achse $(\cos \alpha, \sin \alpha, 0)$. Die Rechtsachse des Quadrats können Sie als Kreuzprodukt der berechneten Hochachse und der Normalen berechnen. Vergessen Sie nicht, die Richtungen zu normalisieren und das Quadrat mit den Seitenlängen 2 `world_radius[0]` zu erzeugen.

Vertices erzeugen Sie im Geometry Shader mit der GLSL-Funktion `EmitVertex`, welche Sie immer dann aufrufen, wenn Sie alle Ausgabevariablen für den nächsten Vertex aktualisiert haben. Um den Triangle Strip abzuschließen, rufen Sie die GLSL-Funktion `EndPrimitive` auf.

³Geometry Shader erwarten als Eingabe stets Arrays, selbst wenn diese bei Punkten nur ein Element enthalten.