Karlsruher Institut für Technologie
Lehrstuhl für Computergrafik

Prof. Dr.-Ing. Carsten Dachsbacher
Dipl.-Inf. Christoph Schied

9. Übungsblatt zur Vorlesung Interaktive Computergrafik im SS 2017

Besprechung am **Mittwoch, 26.07.2017**, 10:30 Uhr.

Dieses Aufgabenblatt behandelt Precomputed Radiance Transfer und Spherical Harmonics.

Ziel der folgenden Aufgaben ist die Implementierung von *Image-Based Lighting*, d.h. die Szene soll durch eine Environment Map beleuchtet werden. Wenn dazu der Strahlungstransport vorberechnet und in einer geeigneten Basis gespeichert wird, kann man anschließend die Beleuchtung unabhängig von der relativen Orientierung von Szene und (Umgebungs-)Lichtquelle in Echtzeit auswerten.

Da der Vorberechnungsschritt Ray-Tracing benötigt, wird er auf der CPU durchgeführt. Es wäre theoretisch auch möglich, die Berechnung auf der GPU vorzunehmen; der Mehraufwand für eine GPU-Implementierung ist jedoch in der Regel nicht gerechtfertigt, da es sich um einen *einmaligen* Vorgang zu Programmbeginn handelt.

Aufgabe 1 *Projektion in Spherical Harmonics*

Damit wir die Beleuchtungsberechnung *schnell* auswerten können, müssen wir den Lichttransport in einer geeigneten Funktionsbasis durchführen. Üblicherweise verwendet man dafür Kugelflächenfunktionen (Spherical Harmonics, SH), da diese eine Orthonormalbasis bilden. Letzteres bedeutet in der Praxis, dass wir ein (kompliziertes) Integral als ein einfaches Skalarprodukt schreiben können.

Projizieren Sie zunächst die Environment Map in Spherical Harmonics. Vervollständigen Sie dazu in `renderers.cpp` die Funktion `precompute_lighting()`, so dass die Koeffizienten in `sh_coefficients_light` gespeichert werden. Beachten Sie, dass Sie *drei* Koeffizienten berechnen müssen, jeweils einen für den Rot-, Grün- und Blaukanal. Projezierien Sie hierfür die vorberechnete zufällige Richtung in die Environment Map.

Hinweis: Beachten Sie, dass wir die SH-Koeffizienten nach jeder Drehung der Lichtquelle neu berechnen. Aus der Vorlesung wissen Sie, dass man auch die SH-Basisfunktionen rotieren kann, um die einmalig berechneten Koeffizienten wiederzuverwenden. Wann ist es sinnvoller, dies umzusetzen?

Aufgabe 2 *Berechnung der Transferfunktion*

Der nächste Schritt ist festzulegen, wie der Strahlungstransport modelliert werden soll. Zunächst gehen wir von diffusem Transport ohne Verschattung aus; damit ergibt sich für die Transferfunktion (Position x, Normale n, Richtung ω):

$$T_x(\omega) = \frac{1}{\pi} \langle n, \omega \rangle^+$$

wobei $\langle \cdot, \cdot \rangle^+$ das nach unten begrenzte Skalarprodukt ist, d.h. $\langle a, b \rangle^+ = \max\{0, \langle a, b \rangle\}$.

Implementieren Sie in `renderers.cpp` die Funktion:

```
static float
transfer(const glm::vec3 & dir,
         const glm::vec3 & vertex,
         const glm::vec3 & normal,
         BVH &bvh);
```

welche den Radiance-Transfer in Richtung `dir` für eine Oberfläche an Position `vertex` mit Normale `normal` auswertet.

Aufgabe 3 *Projektion der Transferfunktion*

Zur Projektion der Transferfunktion müssen wir pro Vertex x das folgende Integral auswerten:

$$t_{x,i} = \int_\Omega T_x(\omega) y_i(\omega) d\omega,$$

wobei y_i die i-te, reelle (SH-)Basisfunktion ist. (Wir verwenden den linearen Index $i = l(l+1)+m$ für Bandindex l und Quantenzahl m.)

Im Programm sollen Sie in der Methode `compute_prt()` dieses Intergral mittels Monte-Carlo-Methoden berechnen:

$$t_{x,i} \approx \frac{4\pi}{N} \sum_{k=0}^{N-1} T_x(\omega_k) y_i(\omega_k),$$

wobei N zufällige, uniform verteilte Richtungsvektoren mit Einheitslänge, ω_k, benötigt werden. Sowohl um die Berechnung zu beschleunigen, als auch um Rauschen durch Varianz zu verringern, werden im Code bereits solche zufällige Richtungen vorberechnet und in `dir_table[]` tabelliert. Außerdem werden für die tabellierten Richtungen auch die SH-Basisfunktionen vorberechnet. Die i-te Basisfunktion für die k-te Richtung erhalten Sie mit dem Ausdruck `sh_coeff_dir[k][i]`.

Wenn Sie sowohl die Transfer- als auch die Beleuchtungskoeffizienten richtig berechnet haben, sollten Sie nun ein korrektes Bild (ähnlich dem Screenshot am Anfang des Aufgabenblatts) sehen können.

Aufgabe 4 *Verschatteter Transfer*

Nun wollen wir auch verschatteten Transfer berechnen, d.h. unsere Transferfunktion soll nun auch die Sichtbarkeitsfunktion V enthalten:

$$T_x(\omega) = \frac{1}{\pi} \langle n, \omega \rangle^+ V(x, \omega)$$

Die Sichtbarkeitsfunktion hat den Wert 1 genau dann, wenn der Strahl von x in Richtung ω *keinen* anderen Teil der Szene schneidet, ansonsten hat Sie den Wert 0.

Ändern sie Sie in `renderers.cpp` die Funktion `transfer()` so ab, dass auch die Verschattung gemäß obigem Ausdruck beachtet wird. Zur Prüfung der Sichtbarkeit können Sie die Hüllkörperhierarchie `bvh` verwenden.

Framework

Wir stellen für jedes Übungsblatt ein Framework bereit. Das Framework nutzt C++ 11 und wird unter Linux getestet. Es ist allerdings auch unter Windows mit Visual Studio 2013 lauffähig. Die Datei `Kompilieren.txt` enthält Informationen darüber, wie sie das Framework kompilieren.

Hinweis: Wenn sie Visual Studio verwenden sollten Sie das Projekt im Release oder RelaseWithDebug starten um die Ladezeiten zu reduzieren.