Übungen zur Vorlesung "Architektur und Programmierung von Grafik- und Koprozessoren"
Übungsblatt 9

Sommersemester 2019

9 GPU Programmierung

Aufgabe 9.1

Zur Lösung dieser Aufgabe benötigen Sie einen PC mit Grafikkarte (die meisten Grafikkarten, z. B. auch integrierte Grafik-Chips, sollten für diese Aufgabe ausreichen) sowie einen Webbrowser, der den Standard WebGL (`https://www.khronos.org/webgl/`) unterstützt.

Für die Aufgabe verwenden Sie das Tool Shadertoy (`https://www.shadertoy.com/`), mit dem Sie ohne aufwendiges Setup Fragment Shader entwickeln können. Erzeugen Sie dazu über das Benutzer-Interface einen neuen Shader und kopieren Sie den Quelltext des zur Aufgabe gehörenden Gerüstprogramms dort hinein. Shadertoy Programme entwickelt man in einer GLSL sehr ähnlichen Sprache.

In der Vorlesung und auch den Übungen haben wir für diffuse Reflektion bisher immer das sehr einfache Lambertsche Modell angenommen. Sie sollen nun das diffuse Reflektionsmodell implementieren, das Burley als Teil von *Disneys Principled BRDF* vorschlägt [1]. Disneys Principled BRDF ist ein empirisches Beleuchtungsmodell, das in Teilen physikalisch plausibel ist, und das in Teilen dadurch motiviert ist, dass Modellierer Parameter intuitiv manipulieren können.

Das diffuse Modell von Burley nimmt eine Mikrofacettenverteilung an. Mikrofacetten reflektieren das meiste Licht entlang der Richtung $\vec{H} = \frac{\vec{h}}{||\vec{h}||}$, wobei $\vec{h} = \vec{\omega_i} + \vec{\omega_o}$ und $\vec{\omega_i}$ sowie $\vec{\omega_o}$ Einheitsvektoren zur Lichtquelle sowie in Reflektionsrichtung sind. Mikrofacetten werden über die Oberflächenrauheit $\sigma \in [0..1]$ modelliert. Burleys Modell sieht eine Fresnel Komponente vor: schaut man in flachem Winkel auf die Oberfläche, wird mehr Licht in die Betrachtungsrichtung reflektiert, als wenn man von oben auf die Oberfläche schaut. Im Modell wird die Schlick Approximation für Fresnel Reflektion verwendet, sodass sich für die reflektierte *Farbe* (Basisfarbe × Strahlstärke) ergibt:

$$f_d = \frac{\text{BaseColor}}{\pi}(1 + (F_{D90} - 1)(1 - cos\theta_i)^5)(1 + (F_{D90} - 1)(1 - cos\theta_o)^5), \quad (1)$$

wobei $F_{D90} = 0.5 + 2cos\theta_h^2\ \sigma$. θ_i und θ_o sind definiert wie in der Vorlesung und θ_h bezeichnet den Winkel zwischen der Oberflächennormale und dem *Halbvektor* $\vec{H}$.

Erweitern Sie das Gerüstprogramm, indem Sie Burleys Beleuchtungsmodell für diffuse Reflektion in der dafür vorgesehenen Funktion implementieren. Als Basisfarbe nehmen Sie RGB = $\{0.8, 0.8, 0.8\}$ an.

Aufgabe 9.2

Erklären Sie, was das Threading Modell neuerer NVIDIA GPUs vom Multi-Threading Modell moderner CPUs unterscheidet. Strukturieren Sie Ihre Antwort gemäß der nachfolgenden Überlegungen:

1. Welche Funktionseinheit (Hardware oder Software) ist für das Scheduling von Threads auf der jeweiligen Architektur zuständig?
2. Werden eventuell Threads synchron ("lockstep") bearbeitet? Wenn ja, auf welcher Architektur? Welche Konsequenzen hat das für den Programmablauf? Wie vergleicht sich dieses Ausführungsmodell mit SIMD Architekturen?
3. GPUs verwalten viele Threads gleichzeitig. Häufig sind zur selben Zeit mehr Thread Gruppen aktiv, als es Streaming Multiprozessoren gibt, sodass die Scheduling Einheiten den Gruppen von Threads wechselnd Zeitfenster für ihre Berechnungen zuweisen müssen. Unter welchen Gesichtspunkten wird zwischen den Thread-Ausführungen hin und her geschaltet?
4. Das Umschalten zwischen Gruppen von Threads geht auf GPUs besonders schnell. Warum ist das der Fall?

Aufgabe 9.3

Zur Lösung dieser Aufgabe benötigen Sie einen PC mit einer CUDA-kompatiblen GPU. Ob Ihre GPU mit CUDA kompatibel ist, können Sie hier nachschauen: `https://www.geforce.com/hardware/technology/cuda/supported-gpus`. In den Kursräumen des Lehrstuhls stehen begrenzt PCs mit CUDA-kompatiblen GPUs zur Verfügung.

Implementieren Sie Matrixmultiplikation zweier Matrizen A und B mit Dimensionen (# Zeilen $\times$ # Spalten) $N \times M$ sowie $M \times N$ mit CUDA. Dazu verwenden Sie das beigefügte Gerüstprogramm. Ihr paralleles GPU Programm verwendet zur Berechnung ein zweidimensionales Gitter der Größe $N \times N$, sodass jedem Matrixelement aus der Ergebnismatrix C ein einzelner Thread zugeordnet wird. Jeder Thread berechnet das ihm über die 2D Thread ID zugeordnete Skalarprodukt aus Zeilen- und Spaltenvektor. Das Gitter teilt sich in Blöcke der Größe 32×32 auf. Sie dürfen vereinfachend annehmen, dass N und M ganzzahlige Vielfache von 32 sind. Die Berechnungen führen Sie direkt im *Globalen Speicher* der Grafikkarte aus.

b.)
Optimieren Sie nun Ihr Programm aus Aufgabenteil **a.)**, indem Sie das Skalarprodukt aus Zeilen- und Spaltenvektor im *Shared Memory* durchführen. Bei der Berechnung des Skalarprodukts iterieren Sie über Blöcke der Größe 32×32 und schreiben die zugehörigen Matrixelemente in ein Shared Memory Array. Da die Blockgröße im Gitter ebenfalls 32×32 beträgt, ist jeder Thread im Block für ein Matrixelement zuständig. Führen Sie nun *stückweise* Skalarprodukte innerhalb der Blöcke durch. Sobald Sie einen Block abgearbeitet haben, iterieren Sie "nach rechts" (Matrix A) bzw. "nach unten" (Matrix B). Dabei akkumulieren Sie die Ergebnisse der lokalen Skalarprodukte und weisen die Summe zum Schluss dem Matrixelement in der Ergebnismatrix C zu, für das der jeweilige Thread zuständig ist. Bedenken Sie, dass Threads, die Operationen im Shared Memory durchführen, synchronisiert werden müssen.

Literatur

[1] Brent Burley. Physically-Based Shading at Disney. Technical report, Walt Disney Animation Studios, 2012.

Das Übungsblatt wird am 04.07.2019 besprochen.