Übungen zur Vorlesung "Architektur und Programmierung von Grafik- und Koprozessoren"
Übungsblatt 5

Sommersemester 2019

5 Pointer Jumping und Ray Tracing

Aufgabe 5.1

Die *Präfixsumme* der Folge $(a_n) = a_1, a_2, ..., a_n$ ist definiert als

$$\begin{aligned} p_1 &= a_1 \\ p_2 &= a_1 + a_2 \\ p_3 &= a_1 + a_2 + a_3 \\ p_n &= a_1 + ... + a_n \end{aligned}$$

a.) Formulieren Sie einen *seriellen* $O(n)$ Algorithmus, der als Eingabe ein Array mit n Elementen erhält und für diese die Präfixsumme berechnet. Das Ergebnis darf im selben Array wie die Eingabe gespeichert werden.

b.) Mit dem parallelen Algorithmus PointerJumping aus der Vorlesung lassen sich ebenfalls Präfixsummen bestimmen. Formulieren Sie den Algorithmus entsprechend um. Wie in der Vorlesung erhält der Algorithmus als Eingabe ein Array, das für jeden Knoten in einem Wald wurzelgerichteter Bäume dessen Vorgänger speichert. Ihr Algorithmus erhält außerdem ein Array, das jedem wurzelgerichteten Baum eine Folge (a_n) zuordnet. Der Algorithmus berechnet die Präfixsummen dieser Folgen. Lautet die Eingabe etwa

	0	1	2	3	4	5	6	7	8	9
S	0	0	1	2	3	5	5	6	7	8
(a_n)	1	1	1	1	1	2	2	2	2	2

dann lautet die Ausgabe:

	0	1	2	3	4	5	6	7	8	9
S	0	0	0	0	0	5	5	5	5	5
(p_n)	1	1	2	3	4	2	2	4	6	8

Ist diese Methode zur Bestimmung der Präfixsumme kosteneffizient?

Aufgabe 5.2

Im Gerüstprogramm zu dieser Aufgabe finden Sie einen kleinen 2D Ray Tracer, den Sie erweitern sollen.
Um das Gerüstprogramm zu übersetzen, benötigen Sie das Cross Platform Build Tool *CMake* (`https://cmake.org/`). Mit CMake lassen sich Projekte organisieren, die aus mehreren *compilation units* bestehen, welche anschließend gelinkt werden. Mit CMake empfehlen sich sogenannte *out-of-source builds*: legen Sie ein Verzeichnis, z. B. als Unterverzeichnis des Gerüstprogramms, an. Dieses können Sie z. B. `build` nennen. Rufen Sie nun *von dort aus* das CMake Kommandozeilenprogramm auf und übergeben Sie den Ordner, in dem sich die Datei `CMakeLists.txt` befindet, als Kommandozeilenparameter, also mit `bash` etwa:

```
mkdir build
cd build
cmake ..
```

Platformspezifische Parameter lassen sich über den *CMake Cache* setzen. Diesen können Sie etwa mit dem Tool `ccmake` beeinflussen, oder durch direktes Editieren der Datei `CMakeCache.txt` im Verzeichnis `build`. Diese Parameter können dem Programm `cmake` auch direkt auf der Kommandozeile mit übergeben werden, z. B.:

```
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_CXX_FLAGS="-std=c++11"
```

a.)
Erweitern Sie den 2D Ray Tracer aus dem Gerüstprogramm. Implementieren Sie zunächst das Lichtbrechungsverfahren aus der Vorlesung in der Funktion `refract()`. Das Gerüstprogramm definiert bereits eine einfache 2D Geometrie mit unterschiedlichen Brechungseigenschaften. An den 2D Plots, die das Programm in eine `.pnm` Datei schreibt, können Sie nachvollziehen, ob Sie die Methode richtig implementiert haben.

b.)
Behandeln Sie außerdem den Fall der totalen internen Reflektion. Diesen Fall sollte Ihre Implementierung von `refract()` erkennen und in diesem Fall einen 0-Vektor zurückgeben. Der Ray Tracer geht im Weiteren davon aus, dass totale interne Reflektion aufgetreten ist, wenn `refract()` einen 0-Vektor zurückgegeben hat und ruft in dem Fall die Funktion `reflect()` auf, um einen neuen Richtungsvektor zu erzeugen. Die Funktion `reflect()` ist von Ihnen noch zu implementieren. Was ist im Gerüstprogramm zu ändern, sodass bei der gegebenen Geometrie und den vorgebenen Primärstrahlen totale interne Reflektion auftreten kann?

Bemerkungen: Bei der Implementierung der beiden Funktionen sollten Sie mit einfachen Vektoroperationen auskommen, das Aufrufen tatsächlicher trigonometrischer Funktionen sollte nicht nötig sein. Bedenken Sie außerdem, dass bei den in der Vorlesung behandelten Modellen *alle Vektoren*, also die Normale, der Vektor zum Licht sowie der Richtungsvektor zum Betrachter *Einheitsvektoren* sind, die vom Schnittpunkt mit der Oberfläche *wegzeigen*.

Das Übungsblatt wird am 16.05.2019 besprochen.