Prof. Dr. Reinhard Klein
Dr. Michael Weinmann
Richard Vock, Sebastian Ochmann

Universität Bonn
Institut für Informatik II
02.07.2018

Sommersemester 2018

Einführung in die Computergraphik
Übung 10

Bildverarbeitung im Deferred Renderer

Aufbauend auf dem Deferred Renderer aus der letzten Übung schauen wir uns dieses Mal an wie sich eine klassische (und wichtige) Operationen aus der Bildverarbeitung mittels Shadern realisieren lässt.

Wie wir bereits gesehen haben arbeitet ein Deferred Renderer in mehreren sukzessiven Render-Durchläufen (Render-Passes). Z.B. wurde in unserer Implementation zunächst Geometrieinformation der Objekte in eine Textur gerendert und die Beleuchtungsrechnung in einem weiteren Durchlauf realisiert, der aus der ersten Textur gelesen und in den Ausgabe-Buffer geschrieben hat.

Viele Operationen aus der klassischen Bildverarbeitung wie Glättung, bestimmte Verfahren zur Kantendetektion, Änderung von Helligkeit, Kontrast oder Farbigkeit arbeiten in kleinen lokalen Bereichen auf dem Eingabebild um die Pixel des Ausgabebildes zu berechnen, d.h. zur Berechnung des Farbwertes eines Ausgabe-Pixels an einer bestimmten Stelle wird nur der entsprechende Pixel im Eingabebild und ggf. Pixel in der Nachbarschaft benötigt.

Für Aufgaben dieser Art bieten sich (Fragment-)Shader geradezu an, da sich das Problem in viele sehr kleine, voneinander unabhängige Teilprobleme unterteilen lässt (nämlich eines für jeden Pixel). Als Beispiel implementieren wir eine Bildglättung indem wir unseren Deferred Renderer durch weitere “Smoothing”-Passes ergänzen.

Faltung, Glättungs-Kernel

Bildglättung wird in der Regel durch eine Faltung mit einem sogenannten Kernel realisiert. Hierzu fasst man das Bild als eine zweidimensionale (diskrete) Funktion f auf und den Kernel als zweidimensionale (diskrete) Funktion g. Die diskrete (zweidimensionale) Faltung ist dann definiert als

$$(f*g)[i,j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[m,n]g[i-m,j-n] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[i-m,j-n]g[m,n],$$

wobei f und g hier als unendlich ausgedehnt angenommen werden; in der Praxis muss der Definitionsbereich entsprechend angepasst werden.

Zwei klassische Kernel sind der Box-Kernel und der Gauß-Kernel. Ersterer ist einfach die konstante Funktion, normiert auf die Summe 1. So sieht ein Box-Kernel der Größe 5x5 folgendermaßen aus:

$$\frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Die Werte des Gauß-Kernels entspringen der zweidimensionalen Gauß-Verteilung

$$G(x, y) := \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

Ein Gauß-Kernel der Größe 5x5 sieht (bei geeigneter Wahl von σ und diskreter Integration über die Fläche der Pixel) etwa folgendermaßen aus (siehe hier):

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Wichtig ist in beiden Fällen die Normierung der Summe der Einträge auf 1, damit der Gesamtbetrag der Farbwerte sich durch die Faltung nicht ändert (d.h. dass das Bild nicht insgesamt heller oder dunkler wird). Für eine Anschauung der Faltung siehe auch hier.

Separable Kernel

Ein naiver Ansatz für die Berechnung des Farbwertes für einen Pixel im Ausgabebild bei Verwendung eines Kernels der Größe $n \times n$ wäre nun, alle $n \cdot n$ Farbwerte der Nachbarschaft um den Pixel im Ausgangsbild auszulesen, mit dem entsprechenden Eintrag des Kernels zu multiplizieren und aufzusummieren. Wir können die Komplexität pro Pixel allerdings mit einem einfachen Trick von $n \cdot n$ auf $2n$ reduzieren:

Ein Kernel K (aufgefasst als $n \times n$ Matrix) heißt separabel, wenn ein n-dimensionaler Spaltenvektor v existiert, sodass gilt

$$v \cdot v^T = K,$$

d.h. das dyadische Produkt auf der linken Seite ergibt genau die Kernel-Matrix. Ist dies der Fall, so können wir den gleichen Glättungseffekt erzielen indem wir das Bild zwei Mal in jeweils einer Dimension falten (nämlich einmal vertikal mit v und einmal horizontal mit v^T) anstatt ein Mal in zwei Dimensionen mit K.

Für den Box-Kernel kann man sich v leicht überlegen. Für den Gauß-Kernel wird die eindimensionale Gauß-Verteilung verwendet:

$$G(x) := \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Analog zum zweidimensionalen Fall muss σ, die Kernel-Größe und die Normierung geeignet gewählt werden.

Glättung im Shader

Die Implementation der vorangegangenen Überlegungen in unserem Deferred Renderer ist recht einfach: Anstatt wie zuvor im ersten (Geometry-)Pass in eine Textur und anschließend in den Ausgabe-Buffer zu schreiben lassen wir das Ergebnisbild jetzt zunächst wiederum in eine Textur schreiben. Diese Textur wird dann an ein Shader-Programm übergeben, welches im Fragment Shader die eindimensionale Faltung mit einem (als Textur) gegebenen Filter-Kernels übernimmt. Ein weiterer Parameter dieses Shaders ist ob die Faltung im aktuellen Durchlauf in horizontaler oder vertikaler Richtung durchgeführt werden soll (aus Performancegründen kann man zwei separate Shader verwenden um sich den if-Branch zu sparen).

Wir können dieses Vorgehen mehrfach wiederholen bis die gewünschte Glättungsstärke erreicht ist. Wichtig hierbei ist, dass wir zwei Texturen zur Bildausgabe verwenden, welche immer abwechselnd benutzt werden: Eine zum Auslesen des aktuelles Ergebnisses und eine zum Schreiben des neuen Resultats; anschließend werden sie mit getauschten Rollen weiterverwendet. Im letzten Durchgang wird anstatt in eine Textur in den Ausgabe-Buffer geschrieben. Die untenstehende Grafik verdeutlicht diese Pipeline.

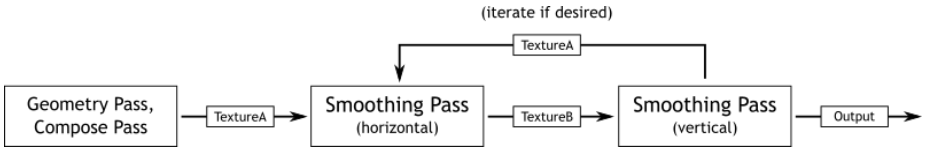

Figure 1: Glättungs-Pipeline