Institut für Visualisierung
und Interaktive Systeme

Dr. Guido Reina Dr. Michael Krone

4. Übungsblatt zur Vorlesung Computergraphik im WS 2017/18

Abgabe am Mittwoch, 22.11.2017 (15:00)
Besprechung am Montag, 27.11.2017 17:30 - 19:00 Uhr

Auf diesem Übungsblatt werden Sie sich mit der Implementierung von Filteroperationen auf Bildern beschäftigen. Das gegebene Programmskelett für dieses Übungsblatt baut auf einer angepassten Version der Bildklasse und der Funktionen des vorangegangenen Übungsblatts auf. Neben dem neuen Namespace `cg::filter::` in dem Sie grundlegende Funktionen für Filteroperationen implementieren bzw. ergänzen werden, wurde das Programmskelett durch einen OpenGL ImageViewer erweitert, den Sie in diesem Übungsblatt verwenden und vervollständigen werden. Für die Bearbeitung der Aufgaben werden keine OpenGL Kenntnisse benötigt – diese werden Sie im Verlauf der Vorlesung aber auch noch erlangen.

Hinweis: Beachten Sie bitte, dass das Programmskelett für die Verwendung von OpenGL auf externe Open-Source-Bibliotheken zurückgreift. In der mitgelieferten Visual Studio Solution sind die Bibliotheken bereits eingebunden. Falls Sie einen anderen Compiler oder ein anderes Betriebssystem verwenden wollen, müssen Sie sich selbst um die Einbindung folgender Bibliotheken kümmern:

- glfw: http://www.glfw.org/
- glad: https://github.com/Dav1dde/glad
- imgui: https://github.com/ocornut/imgui

Das Programm-Skelett sollte bereits fehlerfrei kompilieren und ausführbar sein. Sobald Sie das Programm starten, öffnet sich ein Fenster in dem ein Bild und eine simple GUI angezeigt wird. In der GUI können Sie den Pfad zu einem *.ppm* Bild angeben, das Sie anzeigen und filtern möchten. Darüber hinaus bietet Ihnen die GUI die Wahl zwischen CPU- und GPU-Berechnung der Filteroperationen, sowie jeweils eine Auswahl unterschiedlicher Filter mit entsprechenden Einstellungen. Mit dem *Update*-Button wird der ausgewählte Filter auf das Eingabebild angewandt und das Ergebnis direkt im Fenster angezeigt. Im gegebenen Zustand wird jedoch noch keine Filterkonfiguration eine Änderung des Bildes verursachen. (Ausnahme: Die Ausführung des GPU Filters resultiert in einem schwarzen Bild.)

Hinweis: Beachten Sie bitte, dass in diesem Übungsblatt für die Verwendung von GPGPU-Computing mit einem OpenGL *Compute Shader* ein OpenGL Kontext der Version 4.3 oder höher benötigt wird. OpenGL verwendet *Shader*-Programme, in der C-ähnlichen Sprache GLSL geschrieben, für die Programmierung moderner Grafikkarten. Diese Art der Programmierung werden Sie im Verlauf der Vorlesung näher kennen lernen. Für dieses Aufgabenblatt muss nur eine kleine Erweiterungen an einem schon lauffähigen Shader vorgenommen werden.

Aufgabe 1 *Filteroperationen auf der CPU [8 Punkte]*

Für Filteroperationen auf (diskreten) Bildern werden üblicherweise sog. Kernel verwendet. Eine entsprechende Klasse `Kernel` ist in `ImageFilter.h` bereits fertig implementiert und steht Ihnen zur Verfügung. Lesen Sie die Kommentare zu den einzelnen Funktionen der Klasse aufmerksam durch, um die Klasse korrekt zu verwenden!

1. Implementieren Sie in der Funktion `filterImage`, in Zeile 168 der Datei `ImageFilter.h`, die Anwendung des Filter-Kernels auf das Eingabebild. Iterieren Sie dazu über den Bereich des Filter-Kernel und berechnen Sie die Summe der Multiplikationen der Kerneleinträge mit den entsprechenden Bildwerten (siehe Kapitel 1, Folie 148 der Vorlesung). Verwenden Sie die Funktion `offsetImageCoordinates` zur Berechnung der passenden Bildkoordinaten für eine gegebenen Position des Kernels *über* dem Eingabebild (in Bildkoordinaten) und einer gegebenen Position innerhalb des Kernels (Koordinaten relativ zur Mitte des Kernels).

2. Implementieren Sie anschließend die Funktion `buildEdgeDetectionKernel`, in der ein einfacher 3×3 Filter-Kernel zur Kantenerkennung erstellt werden soll (siehe Kapitel 1, Folie 162 der Vorlesung).

3. Um das Ergebnis des Kantenerkennungs-Filters anzuzeigen, müssen Sie nun in der Datei `ImageViewer.cpp` die Funktion `applyCPUEdgeDetection` ergänzen. Erstellen Sie in der Funktion einen entsprechenden Filter-Kernel und verwenden Sie die Funktion `filterImage` um den Filter-Kernel auf das Eingabebild anzuwenden.

4. Für das *Weichzeichnen* eines Bildes verwendet man in der Computergrafik häufig einen Gauß-Filter. Dabei wird für jeden Pixel im Bild seine lokale Nachbarschaft gemäß einer zweidimensionalen Gauß-Glocke der Breite σ gewichtet. Die Funktionen `build2DGaussianKernel`, `build1DHorizontalGaussianKernel` und `build1DVerticalGaussianKernel` verwenden die Funktion `setGaussianValues` zur Berechnung der Einträge eines solchen Gauß-Filters. Implementieren Sie die Funktion `setGaussianValues` mit folgender Formel:

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

   wobei x und y der horizontale und vertikale Abstand vom Zentrum des Kernels in Pixel ist. Normalisieren Sie anschließend die Einträge des Kernels. Was passiert, wenn keine Normalisierung angewendet wird?

5. Ergänzen Sie die Funktionen `applyCPUGaussian2D` und `applyCPUSeperatedGaussian` in `ImageViewer.cpp` analog zur Implementierung von `applyCPUEdgeDetection`, um die Gauß-Filter im Programm verwenden zu können. Für den separierten Gauß-Filter wird zunächst ein eindimensionaler Filter in horizontaler Richtung and anschließend nochmals in vertikaler Richtung auf das Bild angewandt. Welchen Vorteil hat dieses Vorgehen?

6. Ein wesentlicher Aspekt bei der Anwendung von Filtern ist das Verhalten am Bildrand, da dort für einen Teilbereich des Filter-Kernels keine zugehörigen Bildwerte mehr verfügbar sind. In der Funktion `offsetImageCoordinates` in `ImageFilter.h` werden potentielle Zugriffe außhalb des Bildbereiches erkannt und nach unterschiedlichen Strategien auf gültige Koordinaten innerhalb des Bildes abgebildet. Die Strategie `CLAMP_TO_EDGE`, die alle Zugriffe außerhalb des Bildbereiches auf den Rand des Bildes zurück abbildet, ist bereits implementiert. Ergänzen Sie die Strategie `MIRROR`, die Zugriffe am Bildrand spiegelt, sowie die Strategie `REPEAT`, die das Bild in allen Richtungen virtuell wiederholt indem Zugriffe außerhalb des Bildbereiches auf gültige Koordinaten am gegenüberliegenden Bildrand abgebildet werden.

7. Experimentieren Sie mit unterschiedlichen Konfigurationen der CPU-basierten Filter um Ihre Implementierung zu testen.

Aufgabe 2 *Filteroperationen auf der GPU [2 Punkte]*

Für größere Bilder und normale 2D Filter-Kernel sollte Ihnen aufgefallen sein, dass die Berechnungen einige Zeit beanspruchen können. Die Verwendung von separierten Filtern, wie z.B. für den Gauß-Filter der vorherigen Aufgabe, bringt bereits eine merkliche Verbesserung mit sich. Eine weitere, deutliche Reduzierung der Rechenzeit lässt sich durch die Verwendung einer Grafikkarte zur Berechnung der Filteroperationen erreichen. Das Programmskelett liefert bereits alle grundlegenden Rahmenbedienungen dafür. Ergänzen Sie den Compute Shader `seperated_gaussian_c.glsl` um die Anwendung des Gauß-Filters auf das Eingabebild. Beachten Sie, dass hier ebenfalls ein eindimensionaler Filter verwendet wird und der Shader nur eine einzelne Anwendung des gegebenen Filters auf das Bild durchführt. Um den korrekte Verwendung des Shaders und das Setzen der Eingabegrößen müssen Sie sich nicht kümmern.