Prüfungsaufgabe 1

In dieser Aufgabe soll die Berechnung von Histogrammen für Bilder in OpenCL implementiert werden. Wir beschränken uns dabei auf ein Histogramm für die Helligkeit.
Die Helligkeit ***I*** eines Pixels soll aus den RGB-Werten mit folgender Formel berechnet werden:

$$I = 0.299 * R + 0.587 * G + 0.114 * B$$

Die so berechneten Helligkeitswerte liegen im Bereich zwischen 0 und 255. Zur Bestimmung des Histogramms wird eine Statistik der Helligkeitswerte benötigt, d.h. für jeden der möglichen Helligkeitswerte von 0 bis 255 muss die Anzahl der Pixel bestimmt werden, die diesen Helligkeitswert besitzen. Diese Werte sollen in ein gegebenes Feld aus 256 `int`-Elementen eingetragen werden.

Auf der CPU könnte das z.B. folgendermaßen implementiert werden:

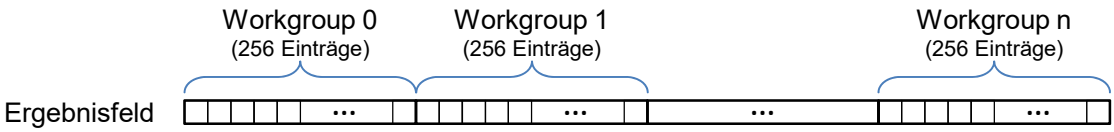
```
struct {
  unsigned char R,G,B;
}Pixel;

void calcHistogram(Pixel img[], int w, int h, int histo[]){
   for (int i=0 ; i<256 ; i++)
      histo[i]=0;

   for (int i=0 ; i<w*h ; i++) {
      Pixel pix = img[i];
      float Y = 0.2126*pix.R + 0.7152*pix.G + 0.0722*pix.B;
      histo[(unsigned char)Y]++;
   }
}
...
int width=…, height=…;       // width and height of image in Pixel
Pixel image[width*height];   // input image
int histogram[256];          // array for result

...
calcHistogram(image, width, height, histogram);
```

Aufgabe a):

Implementieren Sie die Bestimmung der Helligkeitsstatistik eines Bildes in OpenCL. (d.h. die Bestimmung des o.g. Feldes, das die Anzahl der Pixel zu jedem der 256 Helligkeitswerte enthält) (Alternativ dürfen Sie als Eingabe auch anstatt eines Bildes ein Feld vom Typ `unsigned char` verwenden). Gehen Sie dabei wie folgt vor:

- Arbeiten Sie mit Arrays, nicht mit CL-Image-Sampler
- Um eine sinnvolle Parallelisierung zu erreichen, muss die Aufgabe auf eine größere Zahl von Workitems aufgeteilt werden, die alle dann ein oder mehrere Pixel bearbeiten. Wie das Bild dabei geometrisch aufgeteilt wird, spielt nur für simultane Memory-Zugriffe eine Rolle. Ansonsten muss jedes Pixel einfach einmal besucht werden. Es ist deshalb sinnvoll, das Bild als 1-dimensionales Feld der Pixel zu betrachten.
- Implementieren Sie einen Kernel **`calcStatistic`**, der für jedes Pixel die Helligkeit berechnet und die Anzahl der Pixel zu jedem Helligkeitswert (von 0 bis 255) für jede Workgroup separat in einem Feld vom Typ `int` im globalen Memory speichert (siehe Skizze 1):

Prüfungsaufgabe 1



Skizze 1: Ergebnisfeld des Kernels `calcStatistic`, das die Anzahlen der Pixel zu den 256 möglichen Helligkeitswerten für jede Workgroup separat enthält

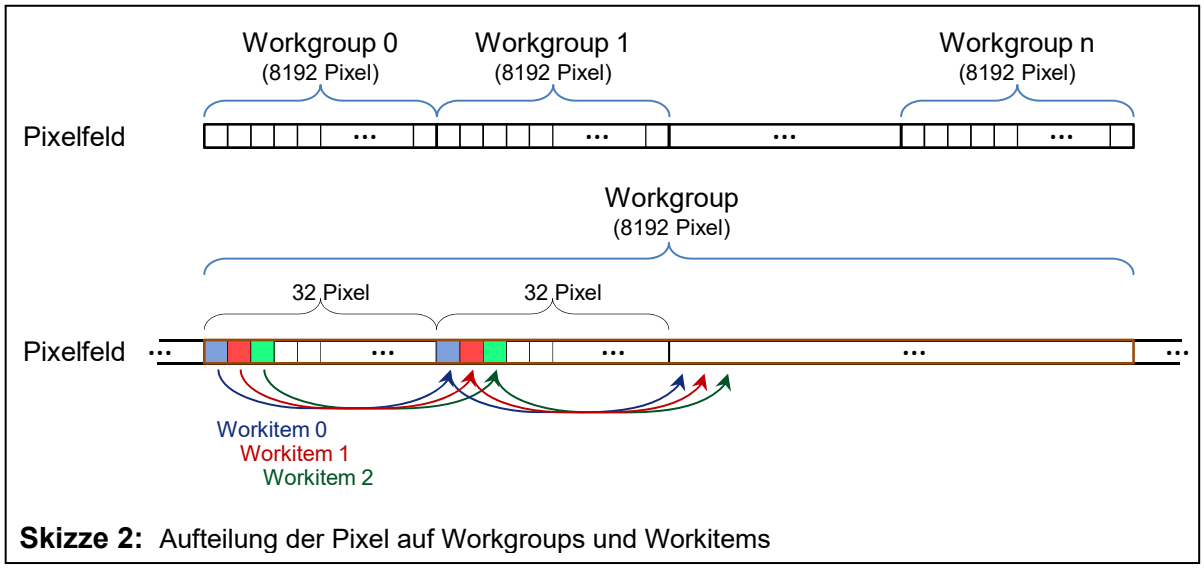
- Der Kernel benötigt als Eingabeparameter das Pixelfeld, dessen Länge und ein Feld vom Typ `int` der Länge `nr_workgroups*256` für die Ergebnisse der einzelnen Workgroups. (`nr_workgroups` ist dabei die Anzahl der Workgroups)
- Bearbeiten Sie in jedem Workitem (in einer Schleife) 256 Pixel. Benutzen Sie eine Workgroup-Größe von 32. (**Achtung**: bitte Hinweis auf Seite 4 beachten) Jede Workgroup bearbeitet damit 32*256= 8192 Pixel. Die globale Worksize erhält man dann, wenn man die Anzahl der Pixel auf das nächste Vielfache von 8192 vergrößert und diese Zahl durch 256 dividiert:

  ```
  global_worksize = (picelcount+8191)/8192*8192/256
                  = (picelcount+8191)/8192*32
  ```

  (Das Feld, das die Pixel enthält, muss nicht vergrößert werden. Sie müssen aber in den Kernels sicherstellen, dass keine Zugriffe über das Feldende hinaus stattfinden.) Die Anzahl der Workgroups ist dann:

  ```
  nr_workgroups = global_worksize/32
  ```

- Damit die Speicherzugriffe auf das Pixel-Feld effizient erfolgen können, soll jedes Workitem innerhalb des Pixelblocks, den die Workgroup bearbeitet, beim LX-ten Pixel beginnen und dann in einer Schleife von da an jedes 32-te Pixel bearbeiten. (LX ist dabei die local_id(0)) (Siehe Skizze 2)



Skizze 2: Aufteilung der Pixel auf Workgroups und Workitems

- Würden mehrere Workitems bei der Berechnung der Statistik die Pixelzahlen im selben Feld aufsummieren, könnte nicht ausgeschlossen werden, dass mehrere Workitems gleichzeitig versuchen, denselben Eintrag zu inkrementieren. Dies würde zu einem falschen Ergebnis führen. Deshalb benötigt jedes Workitem temporär ein eigenes Feld zum Aufsummieren der Pixel-Anzahlen. Benutzen Sie dazu ein zweidimensionales Feld im lokalen Memory

  ```
  local int counts[32][256];
  ```

  Jedes Workitem hat so exklusiv ein Feld mit 256 Elementen zur Verfügung, wenn

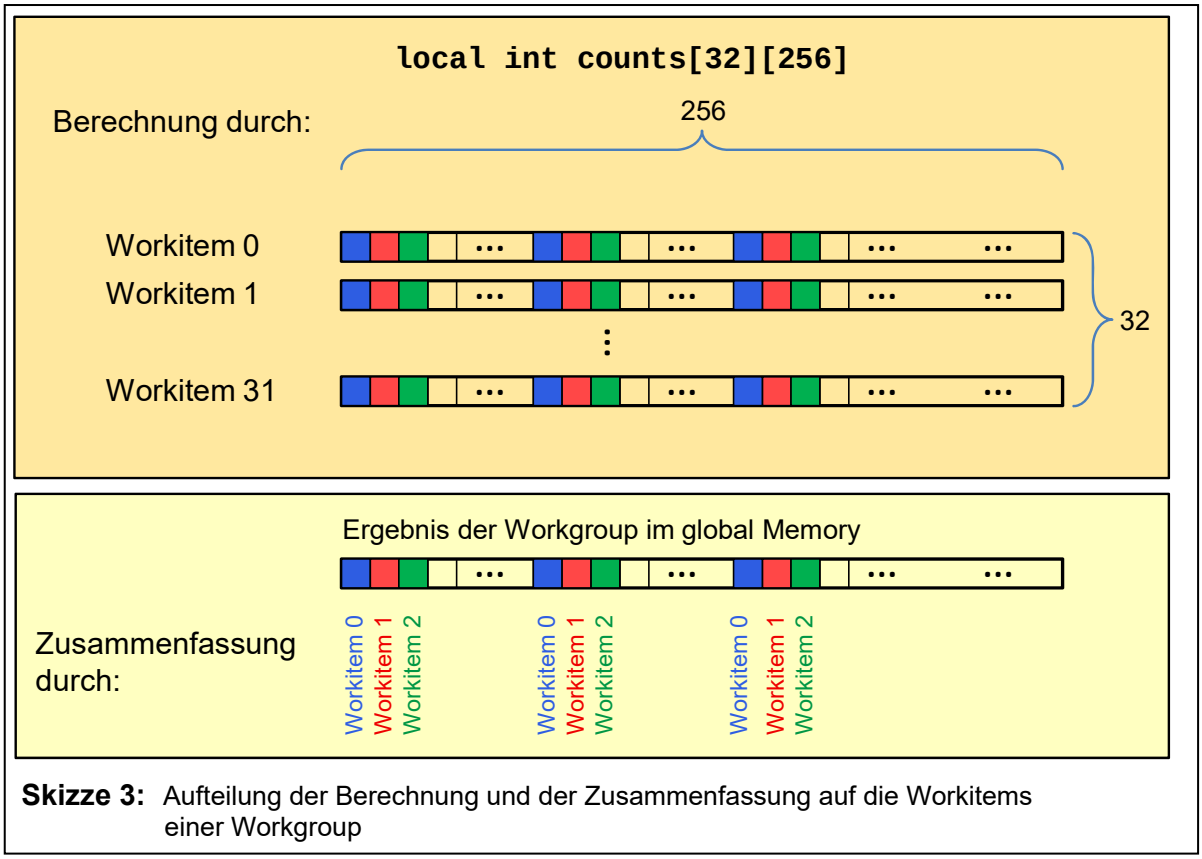
als erster Index die local_id verwendet wird.

Anstatt für jedes Workitem ein exklusives Feld zum Aufsummieren der Pixel-Anzahlen zu verwenden, wäre es auch möglich, mit Atomic-Operations zu arbeiten. Die Aufgabenstellung fordert aber explizit die Lösung mit den exklusiven Feldern.

- Wenn alle 32 Workitems einer Workgroup ihre Pixel bearbeitet haben, müssen die dabei entstandenen 32 Felder zum Gesamtergebnis der Workgroup zusammengefasst und ins globale Memory geschrieben werden. Um effiziente Memory-Zugriffe (sowohl im lokalen als auch im globalem Memory) zu gewährleisten, soll dazu jedes Workitem LX die folgenden Summen berechnen:

$$\sum_{k=0}^{31} counts[k][LX + i*32] \quad \text{für} \quad i = 0,...,7$$

(siehe dazu Skizze 3)



Skizze 3: Aufteilung der Berechnung und der Zusammenfassung auf die Workitems einer Workgroup

- Implementieren Sie einen weiteren Kernel **`reduceStatistic`**, der die Ergebnisse der einzelnen Workgroups addiert und so die Gesamtstatistik für das Bild berechnet. Verwenden Sie dazu eine Workgroup mit 256 Workitems. Addieren Sie in jedem Workitem die Pixelanzahl zu einem Helligkeitswert. Der Kernel benötigt als Eingabeparameter das Ergebnisfeld des vorigen Kernels **`calcStatistic`** (Skizze 1) sowie die Anzahl der dabei verwendeten Workgroups. Der Kernel **`reduceStatistic`** benötigt für das Endergebnis kein weiteres Feld. Die ersten 256 Einträge des Eingabefeldes können durch das Endergebnis überschrieben werden.
- Vor Aufruf des Kernels **`calcStatistic`** muss das Eingabe-Bild ins Grafikkarten-Memory übertragen werden. Zwischen den beiden Kernel-Aufrufen muss das Zwischenergebnis nicht ins mail-Memory übertragen werden. Nach Beendigung des

Kernels **`reduceStatistic`** genügt es, die ersten 256 Einträge des Ergebnis-Buffers ins main-Memory zu übertragen.

Hinweise:

- Sie können die Aufgabe in das Programm *ImageFX* aus den Übungen integrieren.
- **Achtung:** Der Kernel **`calcStatistic`** benötigt für das Feld `counts` 32 KByte lokales Memory pro Compute Unit, was für aktuelle Grafikkarten kein Problem ist. Alte Karten besitzen aber teilweise nur weniger. Die Workgroup-Größe muss dann entsprechend reduziert werden. (z.B. besitzen die Gforce-Karten vonVidia bis Gforce 295 nur 16KByte lokales Memory pro Compute Unit, wovon der Compiler auch noch ein paar Bytes für andere Dinge benutzt. Die Workgroup-Größe muss hier deshalb auf 8 Workitems reduziert werden.)
- In der Aufgabe ist nur die Berechnung der Statistik gefordert. Eine Darstellung des Histogramms ist nicht notwendig.
- Wenn die 4 Bytes (Rot, Grün, Blau, Alpha) eines Pixels als 32-Bit `int`-Wert angesprochen werden, hängt es von der *Endianness* des Prozessors ab, in welcher Reichenfolge die 4 Werte vorliegen (in einem Big-Endian-Prozessor befindet sich der rot-Wert im höchstwertigen Byte, in einem Little-Endian-Prozessor im niedrigstwertigen). Sollten Sie sich über die Werte wundern, könnte das daran liegen. Es ist aber bei dieser Aufgabe nicht gefordert, dass Sie sich mit diesem Problem herumschlagen. Wenn Sie die Werte einfach in der gewohnten Reichenfolge verwenden und Farbkomponenten deshalb möglicherweise falsch zugeordnet werden, ist die Aufgabe trotzdem korrekt gelöst.

Bonusaufgabe:

- Aktivieren Sie in Ihrer OpenCL-Queue das Flag für *out-of-order-execution*, und verwenden Sie Events, um die notwendigen Abhängigkeiten der Daten-Transfers und Kernel-Executions sicherzustellen.
- Erweitern Sie Ihre Lösung so, dass die Anzahl der Pixel, die jedes Workitem bearbeitet (bisher 256), flexibel gewählt werden kann.
- Benutzen Sie die Profiling-Möglichkeiten der Events, um herauszufinden, welche Pixelzahl pro Workitem für Ihre Karte optimal ist. (Benutzen Sie dazu die Differenz zwischen Startzeit des Kernels **`calcStatistic`** und Endzeit des Kernels **`reduceStatistic`**). Steht das Ergebnis in Verbindung mit der Anzahl der Compute-Units, die Ihre Karte besitzt?

Bonusaufgabe 2:

- Implementieren Sie zusätzlich eine Lösung, bei der das Aufsummieren der Pixelzahlen mit Hilfe von Atomic-Operations realisiert wird. Vergleichen Sie die Performance der beiden Lösungen.