Prof. Dr. Günther Greiner, Benjamin Keinert, Matteo Colaianni, Darius Rückert *Erlangen, 28.10.2015*

Partikelsystem (Aufgabenblatt 1)

Aufgabe 1 [4 Punkte] (Interop)

Implementieren Sie zunächst eine Klasse, die das Zusammenspiel zwischen `OpenGL` und `CUDA` vereinfacht. Die Idee ist hier, dass fuer jedes `OpenGL` Buffer Object (hier bei uns nur der `particleBuffer`) - auf welches auch mit `CUDA` zugegriffen wird - auch ein Object der Klasse `Interop` erstellt wird. Zu Beginn soll der Buffer einmalig bei dem Interop Object registriert werden. Anschliessend kann das Buffer Object in den CUDA Address Bereich gemapped und direkt mit einem Kernel modifiziert werden.

a) Erstellen sie eine neue Klasse `Interop`. Legen Sie hierfuer die neuen Dateien `interop.cu` und `interop.h` an (*Hinweis: Beim Hinzufuegen neuer source Dateien muss 'cmake' erneut ausgefuehrt werden*).

b) Implementieren Sie die folgenen Methoden:

- **`void register(int glbuffer);`**
  Hier soll der `OpenGL`-Buffer einmalig registriert werden.
- **`void map();`**
  Hier wird der Buffer gemapped...
- **`void unmap();`**
  ... und hier wieder gelöst
- **`void* getDevicePtr();`**
  Diese Methode gibt den Zeiger auf den gemapped*en* Buffer zurück.

Hinweise:

1. *Das Mapping des Buffer Objects sollte jeden Frame neu durchgeführt werden. OpenGL erlaubt nämlich, daß der Buffer auf der GPU verschoben wird, wodurch sich die Adresse ändert.*
2. *Achten Sie auf eine korrekte Fehlerbehandlung. Verwenden Sie z.B. das checked_cuda Makro aus der particle.cu*
3. *Der Name des OpenGL Buffers der Particles ist particleBuffer.getVBO().*

- http://docs.nvidia.com/cuda/cudaruntimeapi/group__CUDART__OPENGL.html#group__CUDART__OPENGL
- http://docs.nvidia.com/cuda/cudaruntimeapi/group__CUDART__INTEROP.html#group__CUDART__INTEROP

Aufgabe 2 [3 Punkte] (Reset Kernel)

In dieser Aufgabe soll der Startzustand für die Partikel gesetzt werden.

a) Implementieren Sie einen Kernel `resetParticles`, der alle Partikel in einen gueltigen Anfangszustand setzt:
   - Die Partikel sollen in einem *Uniform-Grid* $\in \mathbb{R}^3$ angeordnet sein.
   - Die Ausbreitung in x und z richtung werden durch die Argumente `int x`, `int z` bestimmt.
   - Die Ausbreitung in y Richtung ergibt sich dann durch die Anzahl der Partikel.
   - Die linke untere Ecke des Grids wird durch ein Argument `float3 corner` bestimmt.
   - Der Abstand zwischen den Particle wird durch ein Argument `float distance` bestimmt.

   `resetParticles` soll einmal zu Beginn des Programms ausgefuehrt werden. Zusätzlich kann die Funktion auf einen Tastendruck gelegt werden.

Aufgabe 3 [4 Punkte] (Partikelbewegung)

Implementieren Sie die Funktionalität, um ein Partikel mittels einer anliegende Kraft zu bewegen.

a) Erweitern Sie die Struktur (`CUDA::Particle`) um eine Variable, die den Impuls $\in \mathbb{R}^3$ darstellt. Erzeugen Sie einige Partikel und initialisieren Sie diese für eine gerichtete Emission (z.B. nach oben).

b) Zunächst soll eine `__device__` Funktion implementiert werden, welche den Impuls und die Position des Partikels mittels explizitem Eulerverfahren integriert.

c) Anschließend wird ein Kernel `integrateParticles` implementiert werden, der die Aufgabe hat, ein Partikel aus dem Speicher zu holen, den Integrator aufzurufen und das Ergebnis zurückzuschreiben

d) Rufen Sie diesen Kernel in jedem `update`-Schritt auf. Testen Sie Ihre Implementierung, indem Sie die Partikel mit beliebigen Impulswerten initialisieren.

Aufgabe 4 [2 Punkte] (Lebenszyklus)

Nun soll ein vereinfachtes Lebenszyklus-Modell für die Partikel implementiert werden.

a) Ergänzen Sie eine Datenstruktur `CUDA::Particle` um eine Variable, mit deren Hilfe Sie die Lebensdauer eines Partikels bestimmen können.

b) Aktualisieren Sie in jedem Schritt die Lebenszeit des Partikels. Wenn das Partikel ungültig wird, soll es neu emittiert werden.

c) Machen Sie den Lebenszyklus des Partikels sichtbar, indem Sie die Farbe abhängig vom Alter setzen.

Aufgabe 5 [4 Punkte] (Rauschen)

Nun soll das Ergebnis durch Pseudo-Zufall verbessert werden.

a) Implementieren Sie die `__device__` Funktionen `noise()`, `noise2D()` und `noise3D()`, welche jeweils Zufallswerte in der entsprechenden Vektorlänge zurückgeben. Verwenden sie `cuRand`

b) Stören Sie mit diesem Rauschen die Startposition beim Eintritt in einen neuen Partikelzyklus.

c) Implementieren Sie die `__device__` Funktion `distractDirection3D`, um Richtungen zu stören (*Hinweis: Hemisphere-Sampling*). Stören Sie damit den initialen Impuls am Beginn eines Lebenszyklus.

- http://docs.nvidia.com/cuda/curand/#axzz3prA8KWtM
- http://aresio.blogspot.de/2011/05/cudarandomnumbersinsidekernels.html

Aufgabe 6 [3 Punkte] (Stern)

Nun können wir eine kleine Partikelszene zusammensetzen.

a) Die Partikel sollen sich vom Ursprung aus in eine zufällige Richtung bewegen. Nach Ablauf der Lebenszeit sollen sie wieder in den Ursprung gesetzt werden und mit einem neuen Impuls versehen werden.

b) Nun sollen die Partikel alle in eine Richtung ausgestrahlt werden. Diese Richtung soll nun durch `distractDirection3D` gestört werden. Damit kann man z.B. die Emission einer Turbine simulieren.

Viel Erfolg !

Abgabe: Montag, 09.11.2015, vor 24:00 Uhr.