JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

DezSys08: GPGPU

Systemtechnik Labor
5BHITT 2015/16

Stefan Erceg & Martin Kritzl

Version 1.0

Note:

Betreuer: Prof. Borko & Micheler

Begonnen am 8. Januar 2016

Beendet am 22. Januar 2016

Inhaltsverzeichnis

1 Einführung 3
1.1 Ziele 3
1.2 Aufgabenstellung 3
2 Ergebnisse 4
2.1 Nutzung in normalen Desktop-Anwendungen 4
2.2 Rechenintensive Programme 5
2.2.1 Allgemeines 5
2.2.2 Anwendungsfelder 6
2.3 Nutzung in Entwicklungsumgebungen 7
2.4 Nutzung bestehender Programme (Transcompiler) 8
2.5 Inbetriebnahme der praktischen Beispiele 9
2.5.1 Beschreibung der ausgewählten Algorithmen 9
2.5.2 Infos zum Benchmark 10
2.5.3 Benchmark 10
2.5.4 Fazit 12
2.5.5 Aufgetretene Probleme 13
3 Quellen 14

1 Einführung

1.1 Ziele

Die Aufgabe beinhaltet eine Recherche über grundsätzliche Einsatzmöglichkeiten für GPGPU. Dabei soll die Sinnhaftigkeit der Technologie unterstrichen werden. Die Fragestellungen sollen entsprechend mit Argumenten untermauert werden.
Im zweiten Teil der Arbeit soll der praktische Einsatz von OpenCL trainiert werden. Diese können anhand von bestehenden Codeexamples durchgeführt werden. Dabei wird auf eine sprechende Gegenüberstellung (Benchmark) Wert gelegt.
Die Aufgabenstellung soll in einer Zweiergruppe bearbeitet werden.

1.2 Aufgabenstellung

Informieren Sie sich über die Möglichkeiten der Nutzung von GPUs in normalen Desktop-Anwendungen. Zeigen Sie dazu im Gegensatz den Vorteil der GPUs in rechenintensiven Implementierungen auf [1Pkt].
Gibt es Entwicklungsumgebungen und in welchen Programmiersprachen kann man diese nutzen [1Pkt]?
Können bestehende Programme (C/C++ und Java) auf GPUs genutzt werden und was sind dabei die Grundvoraussetzungen dafür [1Pkt]? Gibt es transcompiler und wie kommen diese zum Einsatz [1Pkt]?

Präsentieren Sie an einem praktischen Beispiel den Nutzen dieser Technologie. Wählen Sie zwei rechenintensive Algorithmen (z.B. Faktorisierung) und zeigen Sie in einem aussagekräftigen Benchmark welche Vorteile der Einsatz der vorhandenen GPU Hardware gegenüber dem Ausführen auf einer CPU bringt (OpenCL). Punkteschlüssel:

Auswahl und Argumentation der zwei rechenintensiven Algorithmen (Speicher, Zugriff, Rechenoperationen) [0..4Pkt]
Sinnvolle Gegenüberstellung von CPU und GPU im Benchmark [0..2Pkt]
Anzahl der Durchläufe [0..2Pkt]
Informationen bei Benchmark [0..2Pkt]
Beschreibung und Bereitstellung des Beispiels (Ausführbarkeit) [0..2Pkt]

Links:

https://github.com/ReneHollander/opencl-example

2 Ergebnisse

2.1 Nutzung in normalen Desktop-Anwendungen

Unter Linux-Betriebssystemen konnte man in den letzten Jahren bei folgenden Anwendungen durch Grafikkarten-Beschleunigung profitieren:

- Software für visuelle Effekte (z.B. „Autodesk Flame")
- Videobearbeitungsprogramme (z.B. „DaVinci Resolve")

In der Windows-Welt findet man folgende Anwendungen mit integrierter GPU-Unterstützung:

- Videobearbeitungsprogramme
  - z.B. „Liquid" war eines der ersten Windows-Programme, welches durch die Verwendung von GPU unterstützt wurde. Die Hilfe von GPUs wird zum Hintergrund-Rendering und für die Berechnung von bestimmten Effekten verwendet.
- ZIP-Programme (z.B. „WinZip")

Allgemein hat sich bewiesen, dass GPUs bei folgenden Desktop-Anwendungen Unterstützung bieten:

- Video-Player (z.B. „VLC Media Player")
- Bildbearbeitungsprogramme (z.B. „Adobe Photoshop")
- 3D-Visualisierungsprogramme
- Datenkonvertierungsprogramme

[1]

2.2 Rechenintensive Programme

2.2.1 Allgemeines

Vor allem die hohe Parallelität von Grafikkarten kann bei einer aufwändigen Berechnung zu einem enormen Vorteil werden. Dies kommt aufgrund der wesentlich höheren Anzahl von Kernen, einer GPU, im Gegensatz zu einer CPU zu Stande. Somit kann jeder Kern einen Algorithmus ausführen, während die übrigen Kerne andere Aufgaben erledigen. [3]

Ein weiterer Vorteil bei Verwendung eines passenden Frameworks z.B. OpenCL, stellt die Verteilbarkeit der Aufgaben auf verschiedene Nodes dar. Dabei können Teile von Aufgaben einem bestimmten Node zugewiesen werden, der die Berechnung durchführt und anschließend die Ergebnisse rückmeldet. [5]

Zur Steigerung ist es notwendig, so wenig wie mögliche Verzweigungen im Programmverlauf zu verwenden. Ebenso sind einige spezialisierte Funktionen vorhanden, die den Standardfunktionen vorgezogen werden sollten. Zum Beispiel in `OpenCL` eher die Funktion `exp(2)` statt `pow(2,x)`. [3]

2.2.2 Anwendungsfelder

2.2.2.1 Biologie

- Ermittlung der kürzesten Strecke zwischen mehreren Punkten im Themengebiet der Genetik

2.2.2.2 Meteorologie/Physik

- Verteilungen von n Objekten unter definierten Voraussetzungen im Raum
- Simulation und Modellierung von Tsunamis
    - Wasser ist Ansammlung von Objekten
    - Änderung eines Punktes, bewirkt Änderung auf alle anderen
    - Äußere Einflüsse wirken auf alle Punkte
    - Massive Parallelität
- Astrophysik
    - Berechnen des Magnetfeldes der Sonne

2.2.2.3 Medizin

- Undersampling
    - Visualisierung von Röntgenbildern verbessert
- 3D-Ultraschall
    - Aufnahme von sehr vielen Bildern verarbeiten
- BioChemie
    - Simulationen, die bei kleiner Änderung der Parameter eine sehr hohe Änderung hervorrufen

2.2.2.4 Industrie

- Rendern und Encoden von Videos
- Beschleunigen von SQL Anfragen
- Ausgabepreis von Anleihen kalkulieren
- Lokalisierung von Öl- und Gas-Vorkommen
    - Messung sowie Visualisierung von Daten

2.2.2.5 Militärische Nutzung

- Radarsysteme
    - Große Mengen von Bildern verarbeiten

2.2.2.6 Kryptographie

- Verschlüsselung/Entschlüsselung von Daten

[4]

2.3 Nutzung in Entwicklungsumgebungen

- ***OpenCL CodeBench***

  OpenCL CodeBench ist ein Plug-In für Eclipse. Dieses ermöglicht eine vereinfachte Codeanalyse und die Durchführung von Unit-Tests für OpenCL-Projekte. [2]

- ***CUDA***

  CUDA ist eine NVIDIA Architektur, mit der Programmteile parallel durch den Grafikprozessor abgearbeitet werden. Falls man das NVIDIA CUDA Toolkit installiert, wird Nsight Eclipse Edition ebenfalls mitinstalliert. Nsight Eclipse Edition stellt eine IDE dar, welche es ermöglicht bestimmten CUDA Code zu entwickeln und Projekte zu verwalten. Die Programmiersprachen C und C++ werden unterstützt, allerdings existieren auch Wrapper für Perl, Python, Ruby, Java, Fortran und .NET [2, 6]

- ***OpenCL Studio***

  OpenCL Studio verbindet OpenCL und OpenGL in einer IDE zusammen um eine hohe Performance bezüglich rechenintensiven Programmen und Visualisierungsanwendungen zu erreichen. [2]

- ***Anjuta***

  Anjuta stellt eine kostenlose IDE dar, welche für den GNOME-Desktop gedacht ist. Indem SDKs von NVIDIA, AMD oder Intel verwendet werden, können OpenCL-Anwendungen gestartet werden. Anjuta kann für die Programmiersprachen C, C++, Python und Java verwendet werden. [2]

2.4 Nutzung bestehender Programme (Transcompiler)

Möchte man einen bestehenden Java bzw. C/C++ Code in OpenCL- bzw. CUDA-Code umwandeln, ist die Verwendung von Transcompiler notwendig. Somit können bestehende Programme, die in Java bzw. C/C++ geschrieben sind, auf GPUs genutzt werden. Vor der Ausführung auf der GPU sollte man jedoch sicherstellen, dass der Code aus rechenintensiven Ausführungen besteht und parallel ausgeführt werden kann.

Folgende Libraries wurden für die Umwandlung des Codes gefunden:

- ***Rootbeer***

  Diese Library macht es möglich, bestimmten Java-Code in CUDA-Code umzuwandeln. Es ist nicht notwendig, einen GPU-Kernel in CUDA oder OpenCL zu schreiben, da lediglich die Klasse, die den umzuwandelnden Code enthält, ein bestimmtes Interface implementieren muss. Mit Rootbeer werden fast alle Eigenschaften von Java, bis auf einige Ausnahmen, wie z.B. dynamische Methodenaufrufe, unterstützt. Im Github-Repository von Phil Pratt-Szeliga [7] sind ebenfalls einige Examples vorhanden, die bei der Umsetzung helfen sollen. [8]

- ***Aparapi***

  Mit Aparapi, welches von AMD entwickelt wurde, wird eine weitere Library für Java-Entwickler zur Verfügung gestellt, um den Code parallel auf einer GPU ausführen zu können. Zur Ausführung wird in einer Methode aus der „Kernel"-Klasse der umzuwandelnde Code hinzugefügt. Der Code wird daraufhin in OpenCL-Code umgewandelt. Falls der Code nicht auf der GPU ausgeführt werden kann, wird ein Java Thread Pool aktiviert, welches den Code parallel auf einer CPU ausführt. Auch bei dieser Library existieren im entsprechenden Github-Repository [9] einige Examples. [10]

2.5 Inbetriebnahme der praktischen Beispiele

2.5.1 Beschreibung der ausgewählten Algorithmen

Folgende 2 Algorithmen wurden ausgewählt:

- ***Mergesort***

  Beim Mergesort wird eine Liste in mehrere Sequenzen aufgeteilt, welche einzeln sortiert werden. Diese werden anschließend zusammengefügt und daraus erhält man eine komplett geordnete Liste. Es werden 2 Elemente aus der Liste herausgenommen und überprüft, welches kleiner bzw. größer ist. Den parallelisierbaren Teil dieses Vorgangs stellt die Sortierung der einzelnen Sequenzen dar.

  Je nachdem, welche Elemente sortiert werden sollen, wird der Speicher mehr oder weniger eine Rolle spielen. Sortiert man beispielsweise int-Werte, wird dies nicht so sehr ins Gewicht fallen wie das Sortieren von komplexen Strukturen.

  Der Kernel zu der OpenCL-Anwendung ist auf folgender Seite einsehbar: [11].

- ***PI-Berechnung***

  Bei dieser Berechnung wird die sogenannte „Bailey-Borwein-Plouffe"-Formel verwendet. Dabei werden jedoch nicht alle Stellen von Pi berechnet, sondern nur die letzten 9 Stellen. Dies wird durchgeführt, um eine maximale Parallelisierbarkeit zu erreichen. Das Programm wurde in C++ geschrieben, um möglichst wenige Abhängigkeiten zu besitzen.

  In der Anwendung kann man eine „Batch Size" angeben, um die Anzahl der Teilberechnungen zu deklarieren. Diese Teilberechnungen werden schlussendlich zu einem Ergebnis zusammengefasst. Die Berechnung von Pi muss aufgrund der genauen Präzision mit 128 Bit-Integers statt 64 Bit-Integers durchgeführt werden. Intern werden dazu 2 64 Bit-Variablen verwendet, die durch komplexe Algorithmen der Berechnung dienen. Durch diese Erweiterung entstehen sehr rechenintensive Operationen.

  Das Programm kann man sich auf folgender Seite herunterladen: [12].

2.5.2 Infos zum Benchmark

Die Algorithmen wurden mit Laptops bzw. PCs ausgeführt, die folgende CPUs besitzen:

- Kritzl's Laptop: 2x Intel(R) Core(TM) i5-4210H CPU @ 2.90GHz
- Erceg's Laptop: 4x Intel(R) Core(TM) i3-2328M CPU @ 2.20Ghz
- Kritzl's PC: 4x Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz

Folgende Grafikkarten besitzen die Laptops bzw. PCs:

- Kritzl's Laptop: Intel(R) HD Graphics 4600
NVIDIA GeForce GTX 850M
- Erceg's Laptop: Intel(R) HD Graphics 3000
NVIDIA GeForce GT 630M
- Kritzl's PC: NVIDIA GeForce GTX 560

Folgende RAMs besitzen die Laptops bzw. PCs:

- Kritzl's Laptop: 16GB DDR3-RAM
- Erceg's Laptop: 4GB DDR3-RAM
- Kritzl's PC: 16GB DDR3-RAM

Folgende Betriebssysteme besitzen die Laptops bzw. PCs:

- Kritzl's Laptop: Windows 8.1 (64 Bit)
- Erceg's Laptop: Windows 8.1 (64 Bit)
- Kritzl's PC: Windows 7 (64 Bit)

2.5.3 Benchmark

Auslastung der Laptops bzw. PCs, während der Algorithmus auf der GPU berechnet wurde:

- Kritzl's Laptop: ~ 8% CPU; 56MB RAM; 0% Datenträger
- Erceg's Laptop: ~ 8% CPU; 19MB RAM; 0% Datenträger
- Kritzl's PC: ~ 13% CPU; 30MB RAM; 0% Datenträger

Auslastung der Laptops bzw. PCs, während der Algorithmus auf der CPU berechnet wurde:

- Kritzl's Laptop: ~ 97% CPU; 350MB RAM; 0% Datenträger

2.5.3.1 Durchlaufzeiten

Kritzl's Laptop CPU

```
Intel OpenCL 1.2 is ready. Timer: HPET (14.32 MHz)

OpenCL CPU: Intel Core i5-4210H (4 CUs, 2900 MHz)
 Compiling OpenCL kernels ... done.

Calculating 1.000.000.000th digit of PI. 20 iterations.

 Allocated device memory : 335546368 Bytes
 Batch Size              : 20M
 Reduction Size          : 64

 00h 00m 53.184s Batch  1 finished.
 00h 01m 45.408s Batch  2 finished.
 00h 03m 06.989s Batch  3 finished.
 00h 06m 53.075s Batch  4 finished.
 00h 10m 52.553s Batch  5 finished.
 00h 11m 45.564s Batch  6 finished.
 00h 12m 38.170s Batch  7 finished.
 00h 13m 58.393s Batch  8 finished.
 00h 17m 31.767s Batch  9 finished.
 00h 21m 16.664s Batch 10 finished.
 00h 22m 07.202s Batch 11 finished.
 00h 22m 57.571s Batch 12 finished.
 00h 24m 15.850s Batch 13 finished.
 00h 27m 52.349s Batch 14 finished.
 00h 31m 49.103s Batch 15 finished.
 00h 32m 42.763s Batch 16 finished.
 00h 33m 45.925s Batch 17 finished.
 00h 35m 17.254s Batch 18 finished.
 00h 39m 15.394s Batch 19 finished.
 00h 43m 19.514s PI value output -> 5895585A0

Statistics

 Calculation + Reduction time: 2583.480s + 16.013s
```

Kritzl's Laptop GPU

```
NVIDIA OpenCL 1.1 CUDA 6.0.1 is ready. Timer: HPET (14.32 MHz)

OpenCL GPU: NVIDIA GeForce GTX 850M (5 CUs, 901 MHz)
 Compiling OpenCL kernels ... done.

Calculating 1.000.000.000th digit of PI. 20 iterations.

 Allocated device memory : 335546368 Bytes
 Batch Size              : 20M
 Reduction Size          : 64

 00h 00m 03.683s Batch  1 finished.
 00h 00m 07.284s Batch  2 finished.
 00h 00m 13.388s Batch  3 finished.
 00h 00m 30.036s Batch  4 finished.
 00h 00m 46.599s Batch  5 finished.
 00h 00m 50.248s Batch  6 finished.
 00h 00m 53.861s Batch  7 finished.
 00h 00m 59.918s Batch  8 finished.
 00h 01m 16.193s Batch  9 finished.
 00h 01m 32.443s Batch 10 finished.
 00h 01m 36.147s Batch 11 finished.
 00h 01m 39.844s Batch 12 finished.
 00h 01m 46.023s Batch 13 finished.
 00h 02m 02.707s Batch 14 finished.
 00h 02m 19.280s Batch 15 finished.
 00h 02m 22.947s Batch 16 finished.
 00h 02m 26.574s Batch 17 finished.
 00h 02m 32.636s Batch 18 finished.
 00h 02m 48.898s Batch 19 finished.
 00h 03m 05.092s PI value output -> 589558B00

Statistics

 Calculation + Reduction time: 180.952s + 4.099s
```

Kritzl's PC GPU

```
NVIDIA OpenCL 1.1 CUDA 4.2.1 is ready. Timer: RTC (1 ms)

OpenCL GPU: NVIDIA GeForce GTX 560 (12 CUs, 1104 MHz)
 Compiling OpenCL kernels ... done.

Calculating 1.000.000.000th digit of PI. 20 iterations.

 Allocated device memory : 335546368 Bytes
 Batch Size              : 20M
 Reduction Size          : 64

 00h 00m 02.421s Batch  1 finished.
 00h 00m 04.816s Batch  2 finished.
 00h 00m 08.624s Batch  3 finished.
 00h 00m 18.664s Batch  4 finished.
 00h 00m 28.650s Batch  5 finished.
 00h 00m 31.065s Batch  6 finished.
 00h 00m 33.478s Batch  7 finished.
 00h 00m 37.232s Batch  8 finished.
 00h 00m 46.945s Batch  9 finished.
 00h 00m 56.611s Batch 10 finished.
 00h 00m 59.026s Batch 11 finished.
 00h 01m 01.420s Batch 12 finished.
 00h 01m 05.221s Batch 13 finished.
 00h 01m 15.257s Batch 14 finished.
 00h 01m 25.239s Batch 15 finished.
 00h 01m 27.654s Batch 16 finished.
 00h 01m 30.048s Batch 17 finished.
 00h 01m 33.821s Batch 18 finished.
 00h 01m 43.536s Batch 19 finished.
 00h 01m 53.202s PI value output -> 589558B00

Statistics

 Calculation + Reduction time: 111.379s + 1.813s
```

2.5.4 Fazit

Es ist eindeutig zu erkennen, dass die Ausführung auf der GPU um ein vielfaches schneller ausgeführt wurde als auf der CPU. Dabei ist ein Faktor von ca. 14 zwischen den beiden Varianten. Jedoch ist zu sehen, dass die Berechnung der Stellen auf der GPU zu einem falschen Ergebnis geführt hat, was sich auf eine gewisse Ungenauigkeit der Double Precision-Berechnungen zurückführen lässt.

2.5.5 Aufgetretene Probleme

Zu Beginn haben wir uns die Java-Library LWJGL heruntergeladen [13] und diese in der IDE IntelliJ über Maven importiert. Beispielprogramme waren hierbei bereits vorhanden, unter anderem ein OpenCL-Beispiel, welches die Elemente zweier Listen miteinander addiert.

Folgendermaßen wurde in der main-Methode angegeben, ob die Anwendung auf der CPU oder auf der GPU ausgeführt werden soll:

```
List<CLDevice> devices = platform.getDevices(CL_DEVICE_TYPE_GPU);
```

Das generelle Problem bei dieser Aufgabe war, dass die CPU und eine von 2 Grafikkarten bei den jeweiligen Examples nicht gefunden wurden. Sobald wir die CPU als Device angegeben haben und die Anwendung ausgeführt haben, ist eine „EmptyList"-Fehlermeldung aufgetaucht. Wir haben uns nachher zwar den OpenCL-Treiber für die Intel CPU heruntergeladen [14] und installiert, allerdings ist das Problem selbst nach der Installation aufgetreten und die CPU wurde bei den Beispielen nicht erkannt.

Genauso haben wir uns einen Treiber für die interne Intel-Grafikkarte heruntergeladen [15], welche ebenfalls nach der Installation weiterhin nicht erkannt wurde.

Auf einem weiteren Gerät (Kritzl's PC) wurde die CPU von dem GPUPI-Programm erkannt, jedoch wurde die Fehlermeldung „Could not start worker thread" ausgegeben. In der Beschreibung ist erklärt, dass die Fehlermeldung aufgrund einer zu alten CPU ausgegeben wurde und der Benchmark nicht durchgeführt werden kann.

Nach erneutem Versuch das Programm GPUPI auf Kritzl's Laptop auszuführen und die notwendigen Fehlerbehebungen durchzuführen, war es möglich die CPU als Device zu wählen. Dazu war folgender Befehl und anschließender Neustart notwendig:

```
bcdedit /set useplatformclock yes
```

Nun war es möglich den Benchmark ebenso auf der CPU auszuführen. Jedoch konnten andere Implementierungen wie `lwjgl` die CPU noch immer nicht erkennen.

3 Quellen

[1] Peter Knoll (Juni 2012).
Grafikkarte als Software-Beschleuniger [Online].
Available at: http://www.pc-magazin.de/ratgeber/grafikkarte-als-software-beschleuniger-1284905.html
[zuletzt abgerufen am 15.01.2016]

[2] GPGPU.org (2002, 2016).
GPGPU – IDEs [Online].
Available at: http://gpgpu.org/tag/ides
[zuletzt abgerufen am 15.01.2016]

[3] B.Sc. Christof Pieloth (August 2011).
Untersuchung der OpenCL-Programmierplattform zur Nutzung für rechenintensive Algorithmen [Online].
Available at: http://christof.pieloth.org/sites/default/files/documents/Masterprojekt%20-%20OpenCL.pdf
[zuletzt abgerufen am 15.01.2016]

[4] Ralf Leichter (Januar 2011).
Anwendungsfelder massiv parallel programmierbarer Grafikprozessoren im wissenschaftlichen Bereich am Beispiel der Nvidia Fermi Architektur und deren Vorläufer [Online].
Available at: http://winfwiki.wi-fom.de/index.php/Anwendungsfelder_massiv_parallel_programmierbarer_Grafikprozessoren_im_wissenschaftlichen_Bereich_am_Beispiel_der_Nvidia_Fermi_Architektur_und_deren_Vorl%C3%A4ufer
[zuletzt abgerufen am 15.01.2016]

[5] Rene Hollander, Paul Kalauner (Dezember 2015).
GPGPU [Online].
Available at: https://elearning.tgm.ac.at/mod/resource/view.php?id=45929
[zuletzt abgerufen am 15.01.2016]

[6] NVIDIA Corporation (September 2015).
Nsight Eclipse Edition Getting Started Guide [Online]. Available at: http://docs.nvidia.com/cuda/nsight-eclipse-edition-getting-started-guide
[zuletzt abgerufen am 15.01.2016]

[7] Phil Pratt-Szeliga [Github-User "pcpratts"] (Last commit: Juni 2015). Rootbeer GPU Compiler – Java GPU Programming [Online]. Available at: https://github.com/pcpratts/rootbeer1 [zuletzt abgerufen am 15.01.2016]

[8] Jens Ihlenfeld (August 2012). Rootbeer – Java für die GPU [Online]. Available at: http://www.golem.de/news/rootbeer-java-fuer-die-gpu-1208-93795.html [zuletzt abgerufen am 15.01.2016]

[9] Github-User "aparapi" (Last commit: Januar 2016). Aparapi [Online]. Available at: https://github.com/aparapi/aparapi [zuletzt abgerufen am 15.01.2016]

[10] 6 committers (2016). aparapi – API for data parallel Java [Online]. Available at: https://code.google.com/p/aparapi/ [zuletzt abgerufen am 15.01.2016]

[11] Eric Bainville (Juni 2011). OpenCL Sorting – Parallel merge, local [Online]. Available at: http://www.bealto.com/gpu-sorting_parallel-merge-local.html [zuletzt abgerufen am 21.01.2016]

[12] overclockers.at (November 2014). Legends never die: GPUPI [Online]. Available at: https://www.overclockers.at/news/legends-never-die-gpupi [zuletzt abgerufen am 21.01.2016]

[13] Lightweight Java Game Library Project (2012, 2014). Download LWJGL 3 [Online]. Available at: https://www.lwjgl.org/download [zuletzt abgerufen am 22.01.2016]

[14] Robert Ioffe (2014, 2015). OpenCL™ Runtime 14.2 for Intel® CPU and Intel® Xeon Phi™ coprocessors for Windows* (64-bit) [Online]. Available at: https://software.intel.com/en-us/articles/opencl-drivers#phiwin [zuletzt abgerufen am 22.01.2016]

[15] Robert Ioffe (2014, 2015). OpenCL™ Driver for Intel® Iris™ and Intel® HD Graphics for Windows* OS (64-bit and 32-bit) [Online]. Available at: https://software.intel.com/en-us/articles/opencl-drivers#iris [zuletzt abgerufen am 22.01.2016]