

# Massively Parallel Contact Simulation on Graphics Hardware using NVIDIA CUDA

Jens-Fabian Goetzmann

post@jefago.de

**Abstract:** This article describes an approach to use the massively parallel computing capabilities of modern graphics hardware to conduct a physical simulation in order to find good solutions to difficult packing problems. Using graphics hardware is shown to be able to provide significant speedups to some classes of problems while other classes of problems are not able to benefit from the massively parallel execution.

## 1 Introduction

The computing resources available on current generation graphics hardware are enormous. Driven especially by the game market's demand for ever more realistic 3-D graphics, current graphics adapters such as the NVIDIA GeForce 8 series feature theoretical computing performances that are a multiple higher than those of current CPUs ([Gre07], [NVI07]), with an even increasing gap.

NVIDIA CUDA<sup>1</sup> (Compute Unified Device Architecture) is an SDK (Software Development Kit) released by graphics hardware manufacturer NVIDIA with the purpose of making it possible for programmers to accelerate general-purpose computations by using the computational resources available to modern GPUs (Graphic Processing Units). The current 1.0 release was announced in June 2007. It features a specially augmented C language.

CUDA is the first SDK to expose the resources on the graphics hardware for general purposes. Before the advent of CUDA, it was only possible to use the GPU for non-graphics purposes by writing specially crafted shader programs, the programming language of which is specifically designed for graphics applications, and also has severe limitations regarding control flow and memory access.

### 1.1 Specialities of CUDA programs

There are numerous facts that have to be considered prior to writing CUDA programs. First and foremost is the fact that in order to fully take advantage of the processing capabilities

---

<sup>1</sup>Called "CUDA" in the following for the sake of simplicity.

of the GPU, the number of threads run parallelly has to be as high as possible. Furthermore, threads have to be *batched* into equally-sized *blocks*. Threads from the same blocks can be synchronized and have effective means of communication, while threads from different blocks may be run on different multiprocessors and thus can not communicate at all.

CUDA also offers multiple types of memory, all of which have specific advantages and disadvantages regarding speed, accessibility and available amounts. Program data may have to be copied between different memory areas to maximize program efficiency.

The control flow of CUDA programs has to be specially crafted: Because of the SIMD (single instruction, multiple data) design of the GPU multiprocessors, deviations within the control flow of threads executed on a single multiprocessor will lead to serialization of the threads' execution thus losing the advantage of parallel execution. Additionally, recursion is not allowed in CUDA functions; It is also impossible to start another batch of threads from within a parallel execution.

## 2 Packing Problems and Contact Simulation

The objective of the work done for the thesis was to use CUDA to accelerate contact simulation algorithms in order to find good solutions for packing problems. Packing problems are a well-studied class of problems in computer science. They are usually defined as either decision problems ("Given a set of items, do they fit into a specified container?") or as optimization problems ("How many items fit into a given container?" or "What is the minimum container for a given set of items?"). A large sub-class of packing problems are of geometrical type – whether two- or three-dimensional. In these packing problems, the objects to be packed and the container can be described as geometric objects. The objects and the container may have arbitrarily complex shapes and a different number of degrees of freedom.

As most geometric packing problems are NP-complete, an optimal solution is in most cases impossible to find ([BSW]). Thus all approaches for finding good packings compute some approximation to an optimal solution. One possible approach is a physically based *contact simulation*, i.e. a simulation of the objects to be packed as rigid bodies that have a certain mass and can move, collide with each other and the container and exchange impulses. A series of contact simulation steps will in almost all cases lead to a stable state, in which no object overlaps any other object anymore. Thus optimizations can be performed by e.g. putting one more object into the container (in a possibly illegal state) and legalizing the packing using contact simulation.

The problems considered in this case are the two-dimensional packing of circles of the radii 1 to  $n$  in the smallest possible enclosing circle ([Zim05], [ALS06]) and the packing of as many  $200 \times 100 \times 50$ -mm-sized boxes as possible into a trunk, which is modeled by a dense point cloud ([EFRS03], [EFK<sup>+</sup>05]).

### 3 Using CUDA for Contact Simulation

For the problem of circle packing, different parallelization strategies were evaluated.

The first one uses one packing at a time and parallelly executes several randomized improvement trials followed by a series of contact simulation steps. This strategy leads to a quick convergence of the packing to a *stable* packing that cannot be improved anymore by randomized improvement trials.

The second strategy uses multiple independent packings, that are parallelly tried to be improved. The improvement of the different instances is monitored centrally, and if a packing does not improve anymore, it is replaced by a newly generated, random starting packing. This strategy was conceived to be run for a long time, in the hope that eventually a very good packing would result.

The last strategy fully parallelizes the contact simulation of just one packing. However, it turned out that this is not efficient at all, mainly because there is a need for very much inter-thread communication, which is a severe bottleneck.

The problem of trunk packing is incomparably more difficult: The boxes have more degrees of freedom, and a point cloud is a very complicated container when compared to an enclosing circle.

To be able to parallelize the contact simulation for the trunk packing problem, several optimization steps have to be performed: The points are stored in a regular grid. For each box, the grid cells overlapped by the axis aligned bounding box of the box are then searched parallelly and the points are checked for collision with the box. For the collision test of the boxes among each other, the pairs are first filtered by the distance of their centers, before performing a separating axes test ([GLM96]) to check the eligible pairs for collision.

### 4 Results

The evaluation of using NVIDIA CUDA for massively parallel contact simulation has lead to ambivalent results: For the problem of circle packing, an efficient implementation was created that performs about 50 times as fast as a comparable serial implementation. For the complex problem of trunk packing, the conceived implementation performs only slightly better than a comparable serial implementation.

All in all, it could be proved that the enormous computing resources of modern graphics hardware can now be easily used for non-graphics tasks. On the other hand, it has become clear that no problem can be parallelized without thorough consideration of the specialities of CUDA and the GPU, and that some problems can not benefit from the hardware in spite of a highly optimized implementation.

## References

- [ALS06] Bernadetta Addis, Marco Locatelli, and Fabio Schoen. Efficiently packing unequal disks in a circle: A computational approach which exploits the continuous and combinatorial structure of the problem. *Optimization Online*, 1343, 2006.
- [BSW] Tobias Baumann, Elmar Schömer, and Kai Werth. Solving geometric packing problems based on physics simulation. Unpublished Manuscript.
- [EFK<sup>+</sup>05] Friedrich Eisenbrand, Stefan Funke, Andreas Karrenbauer, Joachim Reichel, and Elmar Schömer. Packing a Trunk - now with a Twist! In *ACM Symposium on Solid and Physical Modeling*, pages 197–206, 2005.
- [EFRS03] Friedrich Eisenbrand, Stefan Funke, Joachim Reichel, and Elmar Schömer. Packing a Trunk. In *11th European Symposium on Algorithms*, pages 617–629, 2003.
- [GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996.
- [Gre07] Simon Green. NVIDIA CUDA FAQ Version 1.0. <http://forums.nvidia.com/index.php?showtopic=36286>, 2007.
- [NVI07] NVIDIA Corporation. NVIDIA CUDA Compute Unified Device Architecture, Programming Guide. [http://developer.download.nvidia.com/compute/cuda/1\\_0/NVIDIA\\_CUDA\\_Programming\\_Guide\\_1.0.pdf](http://developer.download.nvidia.com/compute/cuda/1_0/NVIDIA_CUDA_Programming_Guide_1.0.pdf), 2007.
- [Zim05] Al Zimmerman. Al Zimmermann's Circle Packing Contest. <http://www.recmath.org/contest/CirclePacking/index.php>, 2005.