A GPU-based Method for Massive Simulation of Distributed Behavioral Models with CUDA

Ugo Erra
University of Basilicata
ugo.erra@unibas.it

Bernardino Frola
University of Salerno
ber.frola@gmail.com

Vittorio Scarano
University of Salerno
vitsca@dia.unisa.it

Abstract

This work reports the results of a GPU-based approach for the massive simulation of a distributed behavioral model. In this model an agent has a local perception of the world and then it moves by coordinating with the motion of its neighbors. This carries a very high computational cost in the so-called nearest neighbors search. By leveraging the parallel processing power of the GPU and its new programming model called CUDA, we implemented a spatial hashing where a partitioning of the space is used to accelerate the neighbors search. Through extensive experiments, we demonstrate the effectiveness of our GPU implementation when simulating the motion of high-density agent groups.

Keywords: massive simulation, distributed behavioral models, graphics processing unit

1 Introduction and Motivation

Agent-based simulation is a common way to implement autonomous characters or agents to create crowds and other flock-like coordinated group motion. The number of agents involved in such collective motion can be huge, from several hundred birds to millions of fish schooling.

Reynolds (1987), presented the first behavioral model for computer animation applications. In this model every agent, called boid, takes decision using a local behavior model and it moves by coordinating with the motion of a limited number of its neighbors. Then, in order to obtain interactive results each agent must be able to identify efficiently neighbors among all existing agents in the world. This problem was already pointed out by Reynolds as a bottleneck, and suggested spatial hashing as a solution to avoid to the $O(n^2)$ of the brute force approach.

In this work, we present a GPU-based approach for massive simulation of distributed behavioral models by using CUDA (2008) framework for the graphics card NVIDIA G8x series. We adopt the GPU processing power for implementing a uniform data grid to support local perception in the nearest neighbors search. The simulation uses grid cells to keep track of the agents' position in the space and leverage the GPU offloads the sorting to build up the data grid structure to the GPU. The sorting is performed inside the cells to optimize the search and then to quickly obtain information about neighbors for each agent in parallel. Then, the GPU calculate for each agent the steering forces and update its position according to the Reynolds model.

2 The GPU-based Method

In order to avoid the $O(n^2)$ complexity of the neighbors search due to the communication of each agent with every other agent in the world, we adopt a common strategy based on the assumption that interaction of steering behavior drops off with distance. Then, we are interested only to compute efficiently a limited amount of neighbors agents. This assumption alleviates the computational effort required by the neighbors search as well as the difficult to manage dynamic data structures which are not trivial to implement on the GPU.

A video of this work is available at http://isis.dia.unisa.it/projects/behavert/.

Figure 1: Performances with increasing number of agents. On the left, simulation time in fps with and without rendering (we used a simple impostor to render a boid). On the right, computational times in milliseconds broken down into principal phases. We considered the average values on 1000 frames.

Our implementation is inspired by the work on a particle system of Green in Nguyen (2007) which uses a static uniform grid data structure to compute the list of neighbors particle from a given particle. In fact, behavioral models have an important common point with particle systems; each agent is independent and, once computed the neighbors, they can be simulated in parallel.

Then, in order to accomplish this task, a static grid subdivides the world in cubic cells of the same size. Each cell in the world has an unique id. To aggregate in the GPU memory all the agents inside the same cell we assign a hash value to each agent based on its center position. In particular, given an agent position the hash value is the id of its cell. At the end of this step, the GPU performs a radix sort based on id cells. This reordering allows to identify quickly all agents inside the same cell as well as to increase the cache hit rate during neighbors search. In fact, to find all agents within a given region it is sufficient to consider agents inside the cells that overlap a region of interest.

3 Performance Evaluation and Conclusion

We performed a series of experiments in order to measures the performance of our approach. The tests were executed on an Opteron 252 2.6Ghz equipped with 2GB RAM and a GeForce 8800GTS 512MB. All the kernels were written in CUDA 2.1 and the application in C++. In all the tests, the agents are modeled by using the three basic steering behaviors of Reynolds. In the Figure 1, we report a simulation considering only 7 nearest neighbors while each boid steers and a cell size equals to 9. With these values it is possible to simulate about 100K boid agents at 60 fps including both simulation and graphics and up to 1M of boids with interactive performances.

The experiment results showed that this approach can be very effective in implementation of the basic Reynolds model. By exploiting the GPU processing power, we expect in the future that it will be possible to simulate more complex models or integrate features like the obstacle avoidance and the path planning.

References

CUDA (2008). *NVIDIA CUDA Programming Guide 2.0.*

Nguyen, H. (2007). *Gpu gems 3.* Addison-Wesley Professional.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, New York, NY, USA. ACM.