GPU-Accelerated Motion Input Analysis for Motion Reconstruction

Rafael R. Drumond, Esteban Clua
Universidade Federal Fluminense, Niteroi, Rio de Janeiro, Brazil.
rdrumond@ic.uff.br, esteban@ic.uff.br

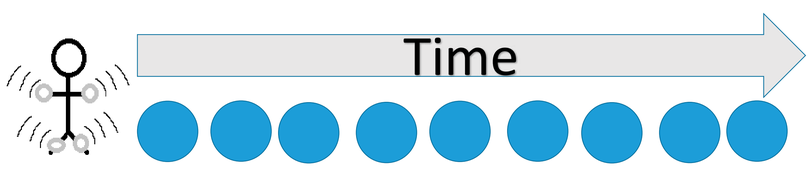
What is this poster about ?

Games and real-life simulations introduced motion capture as an interactive controlling feature which allows humans to reproduce movements that they want to see inside the game. However, reconstructing motion can be slow, especially when it's necessary to look up a huge pre-recorded database to. This work seeks to present a GPU-parallelized algorithm of a previous animation reconstruction method to detect the motion being performed in order to reconstruct it with reduced delay.

Motivation

There are several ways of reconstructing human motion. This works focuses on methods using accelerometers and similar. One of the most recent approach (Tautges, 2011) consists in using a public motion-capture database and comparing it with the information of the real world (from each accelerometer). The motion can be reconstructed by finding the k-nearest neighbors of each accelerometer reading inside the database. After this, the motion sequence is analyzed considering neighbor readings from past frames and building a lazy graph to connect them and solving them to find out what is the current pose of the animated subject. However, simple implementations of motion reconstruction simply aim to find out the type of the motion to be animated (walking, running, crouching and others). This being said, lazy graphs and other algorithms might not be required for these kinds of motion reconstruction and are more suitable for games that require faster computations.

FIGURE 1 – Simplified motion search and reconstruction



Person with Accelerometers **Motion States/Frames from the accelerometers**

At fixed gaps of time the current state of the accelerometers is collected. These states are grouped into chunks that have a fixed amount of states. Each state will be compared to a pre-processed database, by comparing the information of the accelerometers with the information of the frames of the animations on the database.

Instituto de Computação
www.ic.uff.br

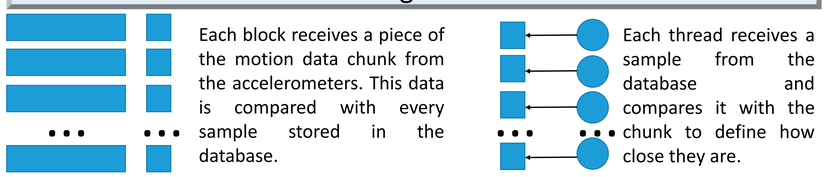
GPU Parallel KNN-Search and Motion Selecting

This work uses a simple GPU parallel KNN-Search.
First a person (or simulation) wears accelerometers to gather data from his/her body and record the information in motion frames. Then this information is compared to the examples in the database to see which pose is the closest to the frame and what type action is being performed. In order to do that we use KNN-Search. After finding out what kind of action it is, an animation is reproduced. KNN-Search is know for being convenient for GPU parallelization. The general way of doing it is by taking just a few steps:

- Each thread calculates the distance from one or a group of samples from the database with the current reading.
- The threads can be divided into blocks, where each block is responsible for one reading (in this case, one of the accelerometers)
- The distance calculations can be ordered by using comb-sort or simply by using min-max atomic reduce operations.
- The first k-elements of the ordered list, represent the k-nearest neighbors. The animations type is defined but the number of neighbors and past animations.

In order to make easier and efficient to compute all the information of all motion-frames we need save memory transfers from CPU to the GPU and vice-versa. So the frames are gathered in chunks big enough to be memory-transfer efficient and small enough to prevent delays.

FIGURE 2 – Block and Thread organization



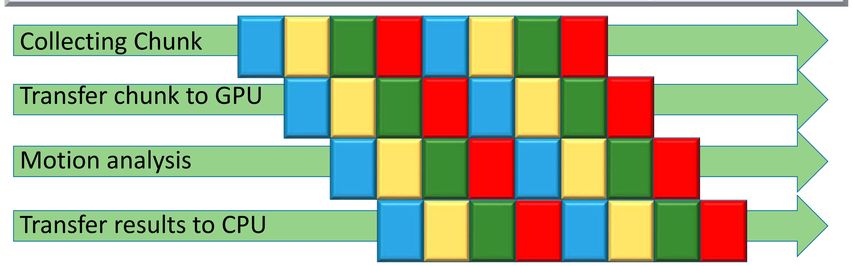
Results

In order to help prevent delays, streams were used. This way, while a group of motion frames are being processed, the next ones would not have to wait to be computed.
The results were achieved by simulating a database and motion-frame. (Note, the time on these tables do not take the delay of recording accelerometer data into account). The table on the left represents the tests with 10 chunks of 60 motion frames and the right has tests with 10 chunks of 100 frames (using accelerometers on wrists and ankles).

Streams	Time taken (ms)	Streams	Time taken (ms)
1	123.843231	1	205.975555
2	62.075134	2	103.096352
3	49.884930	3	82.691101
4	37.572128	4	62.236736
5	***31.022144***	5	***51.590816***
6	34.192738	6	66.961090
NON PARALLEL (CPU EXECUTION)	250,0	NON PARALLEL (CPU EXECUTION)	290,0

As future work, our goal is to test with real accelerometers while using a public motion database. The sizes of the chunks were chosen to be small enough to prevent delays, and big enough to perform less transfers from cpu to gpu.

FIGURE 3 – Motion reconstruction pipe-line (Each color represents a different stream and the actions executed inside it)



Acknowledgements

This project was financially supported by the CAPES-Brazil and CNPq-Brazil. We also would like to thank Rommel Cruz, for helping making this poster.

References

TALBI, E.G. Parallel Local Search on GPU, 2009. Institut National de Recherche en Informatique et en automatique.
Garcia, Vincent, Eric Debreuve, and Michel Barlaud. "Fast k nearest neighbor search using GPU." *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*. IEEE, 2008.
Tautges, Jochen, et al. "Motion reconstruction using sparse accelerometer data." *ACM Transactions on Graphics (TOG)* 30.3 (2011): 18.