CROWD SIMULATION BASED ON FLOCKING BEHAVIOUR ON PARALLEL CUDA PLATFORM

Norhafiza Hamzah*, Norsuzila Yusof & Z.A.Omar

School of Science and Technology,
Universiti Malaysia Sabah, Jalan UMS, 88400 Kota Kinabalu, Sabah, Malaysia
*hafiza@ums.edu.my

ABSTRACT. *This research is focused on flocking behaviour algorithm to simulate the crowd on parallel GPU using CUDA technology. The analysis of frame rates is conducted to compare the crowd simulation on parallel GPU platform and on a single processor. The result shows that the crowd simulation on a parallel GPU platform is 15 frames per second for 16, 384 characters. This result is equivalent with the number of frames per second for crowd simulation on a single processor with 576 characters. Thus, the results demonstrate that crowd simulation is more efficient on the parallel GPU platform especially for the large scale data.*

KEYWORDS: Crowd simulation; Flocking behaviour; GPU computing; CUDA

INTRODUCTION

Crowd simulation is one of focus and became one of the important research areas in computer graphics, virtual reality, social sciences and civil engineering (Jiang *et al*, 2010). Simulation is the movement of large crowds of objects or characters within the same physical environment, and commonly shares the same purpose, and has its own behaviour, while crowd movement includes a number of different agents to move freely in the same environment, targeting, and destination direction (Saboia & Goldenstein, 2012).

Flocking behaviour algorithms first introduced by Craig Reynolds in 1987 and Reynolds introduced the word *boid* to refer to the entity bird (Serrano, 2011). According to Reynolds (2006), the relative motion and the throngs of group and flocking behaviour is often modeled as interacting systems of particles. Flocking behaviour algorithm has three fundamental behaviours: the separation, cohesion and alignment.

CUDA is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU). Scientists and researchers are finding broad-ranging uses for GPU computing with CUDA such as analyze air traffic flow and visualized nanoscale molecules. While being an important tool to enable a faster modeling of problems to a more general parallel machine concept, CUDA abstracted hardware still is very specialized, carrying a different memory model, with huge impacts on the performance of the developed application, depending on its memory access patterns (Passos *et al.*, 2008).

MATERIALS AND METHODS

In this research, crowd simulation algorithm is developed based on flocking behaviour and two additional behaviours such as avoidance and target. The first step to implement the behaviour is to identify the separation of each agent's ability to defend the distance between other agents within a certain radius of the agent environment. Second, the cohesion of the behaviour of the chief determinants or drivers that provide interesting attraction based on the average position of each agent in the agent nearby. Third, the alignment is the steering behaviour that causes entities to attempt to match their directional heading with those of its neighbours. Fourth, avoidance refers to the avoidance of a static target in the global space. The last behaviour is target which attracting the character to a specific location in the global space.

The steps to implement the serial crowd simulation on the parallel GPU platform using CUDA are by identify the *host* (CPU), the *device* (GPU), and which *data* to be parallelized. Next, the data will be sent to the *device* to be distributed as *threads* based on the number of prescribed block dimension. In this research, the number of characters or points used is represented the number of the *thread*. All the threads execute the same instruction. Then, the data will be sent back to the *host* for rendering process. openGL and glut library are used to render the data. The crowd simulation is visualized based on point in OpenGL. At this stage, the interoperability of OpenGL and CUDA is very important to render the data as the position, speed and direction of every point is updated for every timestep. CUDA GraphicsMapResources() is used to map the *vertex buffer object* represented by point position to CUDA memory pointer. This process allows the *vertex buffer object* to be sent to the CUDA *kernel,* and point position is updated based on the defined behaviour. *vertex buffer object* is then released from pointer device using cudaGraphicsUnmapResources() function to execute common OpenGL rendering using glDrawArray() function. The process is shown in Figure 1.

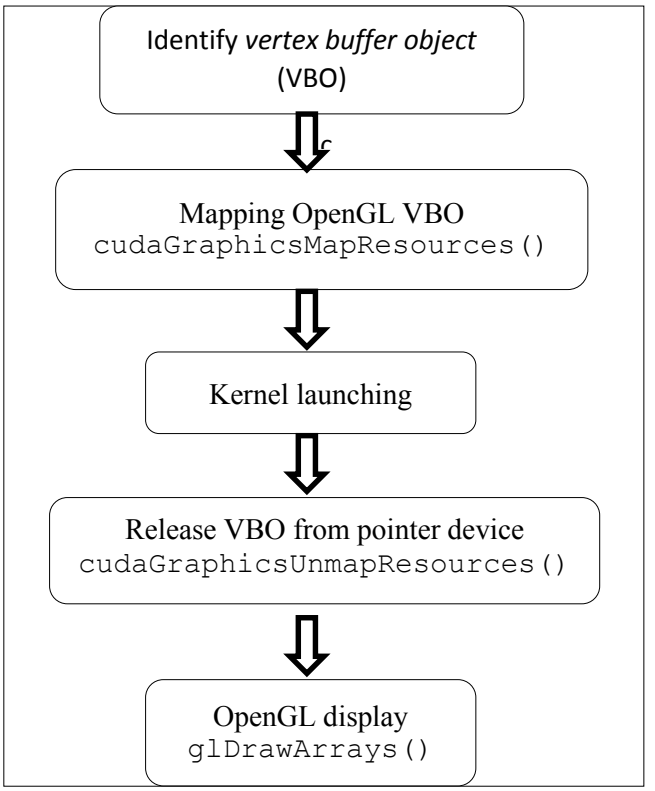

Figure 1: Simulation rendering process through the interoperability of OpenGL with CUDA using *vertex buffer object* (VBO).

RESULTS AND DISCUSSION

The crowd simulation based on flocking behaviour is examined using gDEBugger software on a personal computer with Intel Core i5, Quad CPU processor, 2 GB RAM, and NVIDIA GeForce GT520M Grapics card with 48 CUDA cores. gDEBugger is used to analyzed the frame rates or frames per second for both simulation on parallel GPU and on a single CPU. Then the results were represented in graphs as shown in Figure 2 and Figure 3.

Based on Figure 2, the computational of flocking behaviour is fully computed on the single CPU to obtain the frame rates of the CPU, while the frame rates of the GPU is obtained by sending the data to the device (GPU) and the result is sent back to the host (CPU). The graph shows that the number of frames per second on the GPU is more efficient than the number of frames per second on the CPU. The difference of the frame rates between GPU and single CPU is very significant from the first second. Figure 3 shows the result of frame rates for 16, 384 points. The maximum frame per second is reduced to 15 but still the performance of the GPU was much better than a single CPU.

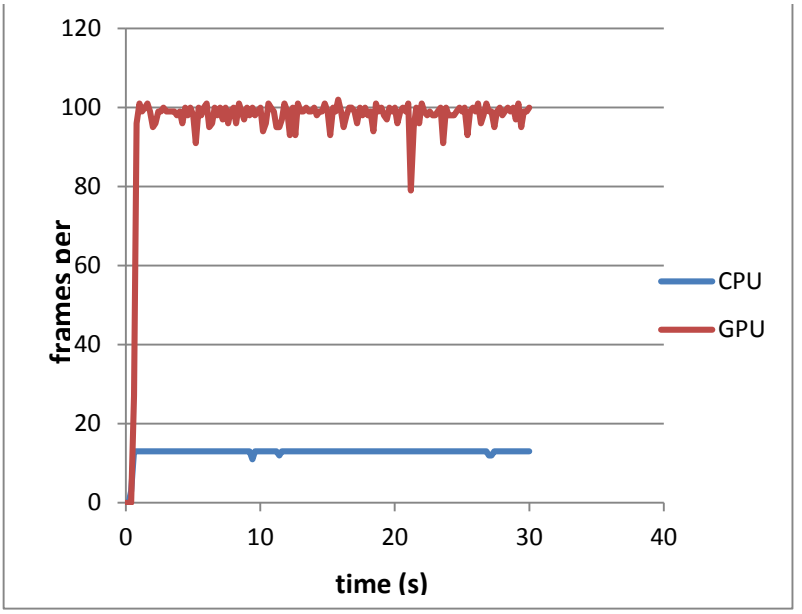

Figure 2: Number of frames per second for GPU and single CPU with 576 points.

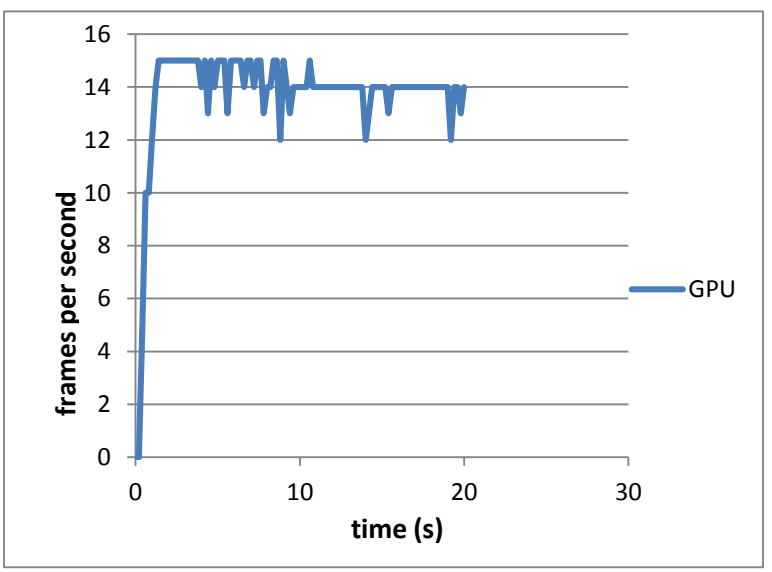

Figure 3: Number of frames per second for GPU and single CPU with 16, 384 points.

CONCLUSION

The comparison of frame rates for simulation on GPU and single CPU to analyze the performance shows that the performance of the GPU implementation by-passed the CPU and sustained interactive frame rates for 16, 384 points. Thus, the execution time of simulation rendering process for a large scale data can be reduced.

REFERENCES

Jiang, H., Xu, W., Mao, T., Li, C., Xia, S. & Wang, Z. 2010. Continuum crowd simulation in complex environments. Computers & Graphics, 34(5): 537-544.

Reynolds, C. 2006. Big fast crowds on PS3. Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames. Boston, Massachusetts: ACM, (pp.113-121).

Saboia, P. & Goldenstein, S. 2012. Vis Comput. Crowd simulation:applying mobile grids to the social force model, pp.1039-1048.

Serrano, M. I. 2011. Flock Implementation for the Blender Game Engine. The Florida State University.

Passos, E., Joselli, M., Zamith, M., Rocha, J., Clua, E., Montenegro, A., Conci, A. & Feijó, B. 2008. Supermassive crowd simulation on GPU based on emergent behavior. Proceedings of the Seventh Brazilian Symposium on Computer Games and Digital Entertainment (SBGames'08), Sciedade Brasileira de Computação, SBC, pp.70-75.