Artificial Intelligence within Games

A review of the Graphics Processing Unit to calculate Finite State Machines

Andrew Jones  
School of Computing Science  
Newcastle University  
Newcastle  
[a.m.jones2@newcastle.ac.uk](mailto:a.m.jones2@newcastle.ac.uk)

**Abstract**- This paper looks at the research into developing artificial intelligence to be implemented onto the GPU for use within a game. GPGPU is explored throughout the paper, starting with languages for GPGPU and what constraints it may have. The use of AI within games in general is explored to give the reader an understanding how modern AI is within a commercial game. As well as looking at very modern research it looks at some earlier techniques for originally exploiting the shader language to create finite state machines.

**Keywords** – GPU, GPGPU, FSM, OpenCL, DirectCompute,

1. Introduction

Artificial intelligence plays an important role within modern day video games and there has been a lot of research put into developing simple but effective AI systems. These systems are effective at what they do however, they are made with a single CPU in mind and don’t offer much parallelism thus, their scalability suffers. Slowly less and less of the CPU’s time is being given to AI [1] however, in some games there is a need for a large number of agents and with a system that suffers in scalability this would not be possible.

The GPU has a parallel architecture and utilizing this would give an enormous performance increase allowing for a large number of agents to be run at once. Over the years, there has been a lot of interest in General Purpose computation on the GPU (GPGPU) and especially for games. This includes using the GPU to calculate, physics, complex mathematic and search algorithms [2]. The GPU, being a graphics processor, was primarily designed with the aim of streaming and processing graphics primitives. This meant that original research and methods had to use graphics libraries as a way of attempting to exploit the GPU for performance benefits [3]. In the last few years, languages such as CUDA and OpenCL have been developed and allow the programmer direct access to the GPU. Which has prompted more interest in the subject however, knowledge of the structure of the GPU architecture is still required to efficiently use the GPU.

In this paper, techniques for implementing AI, especially FSMs, on the GPU are explored and discussed. The paper looks at techniques developed and problems that can arise but looks at them for use within a game environment. This paper will look into what methods of exploiting the GPUs power currently exist, the use of AI within games, what constraints arise when using GPGPU and the implementation of AI onto the GPU.

1. Programming on the GPU

Using the GPU for computations other than graphics rendering is not necessarily a brand new idea however, it has only been recently that languages have been developed specifically for programming the GPU. Originally programmers exploited shader languages such as OpenGL shader language or HLSL that were made for graphics purposes. These allowed programmers to explore the idea of running general purpose computations on the GPU. Using pixel shader’s colour data, it was possible to send the agent’s data into the GPU and process it without the CPU needing to look at it [4]. With this method, over 2 million agents were able to be created and run simultaneously at 50 frames per second.

Now languages have been developed to give the programmer a platform to develop on that allows them direct access to the GPU. There are 3 main languages that are receiving the most attention, NVIDIA’S Compute Unified Device Architecture (CUDA) [5], the Khronos Group’s Open Computing Language (OpenCL) [6] and Microsoft’s DirectCompute [7]. CUDA will be the primary language this paper looks at however, it is worthwhile briefly looking at the other languages for comparison purposes.

CUDA was initially released in June 2007 and has been getting constant support ever since, so in terms of computer science it is relatively new language. In this period of time there has been a lot of research into developing on it as well as refining and optimizing it. Being developed by NVidia, CUDA is made to be used with NVidia graphics cards however, it only newer cards have compatibility with it, 2006 being the earliest generation.

OpenCL was made for use across heterogeneous platforms by the Khronos Group although, it has since been picked up by Apple and now has been adopted by other big companies for use on their hardware. These companies include AMD, NVIDIA and Intel.

DirectCompute, developed by Microsoft is used within their DirectX 11 framework and for compatibility works on DirectX10 GPUs. This gives direct access to the GPU for programmers using the DirectX framework which is very relevant for games programming. It works on multiple different companies’ hardware although as it is part of Microsoft’s framework it requires Windows Vista or newer operating systems to be used.

1. AI Within games

The main aim of a game should be to give the player an enjoyable experience, if a feature takes a lot of resources but doesn’t increase this factor it will normally not be implemented. This constraint is what divides research AI from game AI as most research AI takes up a lot of resources that need to go towards physics, gameplay or graphics in the game. Artificial intelligence plays a key role in games but has been put lower on the priority list for resources for this reason. With the use of GPGPU, this “divide” may be able to be reduced and thus increasing the player’s experience overall. The solutions found by AI research are generally more complex than is needed for video games. A simpler version can be used and give results that are more suited to the games environment. The reason simpler versions are used is because a framerate requirement is needed for video games whereas AI research does not have this constraint and can take as long as it needs to, to calculate the answers. With the use of GPGPU, these complex AI systems may be able to be ran in real time and thus become applicable to games.

Another direction the GPGPU can take AI is instead of having more complex AI, is to have a lot of simple AI agents. One of the foundations of game AI is finite state machines (FSM). A FSM is a powerful yet simple tool for giving an agent in a game the illusion of artificial intelligence [8]. Pac-man is a main example of FSMs within video games where each one of the ghosts has its own FSM to govern its movement within the world and influenced by the player [9]. This successful technique can be modified to emulate a variety of different agents. More complex versions exist that allow for more complex AI but still keeping to the simple modelling system such as hierarchical finite state machines or nondeterministic variants. These form a foundation for a high quantity of artificial intelligence models in video games.

Some games may not require the conventional agents in their games to have complex AI, however almost all games will require some form of pathfinding. Pathfinding is a major part of AI within video games and can be rather resource expensive. One common technique for pathfinding is A\* because it is accurate and can be ran in real time. There has been a relatively high amount of research into using A\* on the GPU. In 2008, Bleiweiss implemented a version of A\* on the GPU [10]. His solution exploits nested data parallelism and proved that its performance scale to be over an order of magnitude compared to CPU. All these ideas seem positive however, there are constraints that arise when using the GPU over the CPU for AI.

1. GPU Constraints

Unlike consoles, personal computers can have a massively wide range of graphics cards in them and this brings constraints to using GPGPU within games. Games aim to cater for a wide audience of players and this wide range of players will have a wide range of computer specifications. Game developers already add options to allow the player to increase or decrease elements of the game so that it runs best on their computer. These options generally only allow the player to modify graphics levels to effect performance and other options to change simple player game interfaces such as controls however, further options may need to be added if more things are calculated on the GPU.

One problem that would arise with using the GPU for processing AI is firstly that it would take up more resources of the GPU thus making the minimum specification for the GPU to be higher. This would lower the range of players that could play the game which no games company would want to happen if avoidable. Although there are options for reducing graphics to allow players with worse GPUs to still be able to play the game, there are rarely if ever options that effect gameplay elements other than difficulty levels. Adding an option that affects AI would be possible, for example you could reduce the maximum number of agents that can be in the game at one time but this could drastically change how one player plays the game compared to another which is not ideal. Changing the maximum number of agents may work for some games that don’t use the agents as key gameplay influences for example simple crowds. This constraint would need to be thought about early in the games development so that these problems don’t potentially cause massive disruptions into the development of the AI such as it needing to be toned down or, even worse, removed from the GPU.

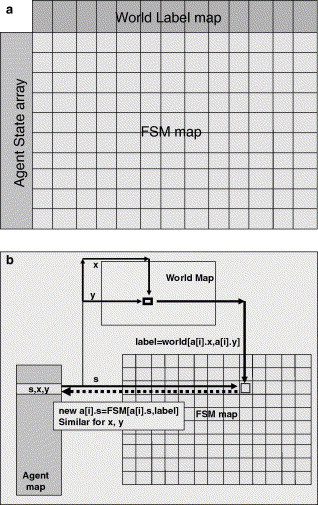
Another problem that could arise is what specific brand of graphics card the player has. As well as needing better graphics cards to be able to run AI on the GPU alongside graphics and other parts of the game, the brand of graphics card effects what language can be used. For example CUDA only works on newer NVidia graphics cards and no other brands. This will dramatically reduce the range of players that would be able to run the code, so the choice of language should be taken carefully. However these problems only fully arise when the system is made for use within a computer game.

Consoles don’t have the problem of each player’s graphics card having different processing power or brand because every one of them is the exact same on the same console. This gives the developer a perfect scope for what they should be able to use and how intensive they can make it. As well as this, console developers such as Sony have started to design their system with GPGPU also on their mind. The Playstation 4’s architecture allows for more communication between the GPU and Memory via the GPU buss which allows the GPU to share some memory with the CPU and memory is a major problem with GPGPU.

1. FSM Using Shaders

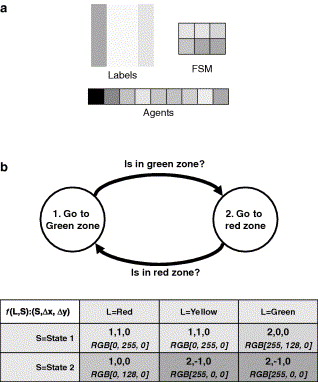
As finite state machines are a very common technique for agent AI, there have been a few different researches into implementing them on the GPU. Even before GPGPU languages had been created people researched into attempting to create a FSM that worked on the GPU. Now that languages exist, more and more researchers are attempting to implement a version of a FSM on the GPU.

Stated previously, before GPGPU languages were created, programmers had to exploit shaders to harness the power of the GPU. In 2005, Rudomín [11] created finite state machine based agents using fragment shaders and was rather successful. Fragment shaders allow the user of several textures and texture lookups and this is the core part of what allowed a FSM to be created. Three maps are created from the textures, firstly a world map is created which is a normal map of the world. This map can contain any information about the world and there can be an arbitrary number of these maps defined by the designer. These maps may be collisions maps, height maps, normals or action maps. The second map is the agent space map, this size of the texture is the number of agents and the colour of the textures map to variables of the agent, for example, RGB = XYZ and A (alpha) = state. The final map is the FSM map. This map is a lookup table where the state of the agent can map to the U axis and the value on the world map at the X and Y coordinate to the V axis as shown in Fig1. Once this mapping has been done, simple finite state machines can be implemented onto it.



**Fig. 1.** How maps link to each other in the fragment shader via textures [11]

An extremely simple finite state machine could be an agent patrolling between two points on a map. For this the agent would have 2 states, one would be moving to the 1st point and then the other state would make the agent move towards the 2nd point. In the FSM table, it would look at what state the agent is in and its position in the world. From these the colour values at the location would indicate the direction in which the agent should move.

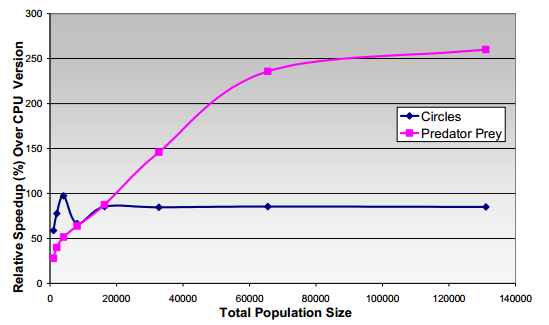


**Fig. 2.** Simple patrol FSM and Maps [11]

Overall this method works and doesn’t use any GPGPU language however, it is very limited in how much data agents can have and there is next to no communication between agents. Rudomín, also shows how hunter and prey can be implemented using this method by adding more world maps and FSM maps that allow the hunter to wander and chase, proving even more complex finite state machines can be implemented with some work even with these limitations.

1. FSM Using CUDA

Once GPGPU languages were developed, research into using them for AI followed quickly and finite state machines were no exception. In 2009, Joselli and Clua managed to implement an entire game on the GPU using CUDA which used FSMs for their agents [12]. They came to the conclusion that even more complex AI should be able to be implemented on the GPU such as fuzzy logic or hierarchical state machines. Richmond, Coakley and Romano created an agent based model using CUDA and compared its speed with that of a CPU [3]. It compares two different models, Circles manages to stay around a 90x speed and the common Predator Prey model converges around 250x faster than the CPU implementation shown in Fig 3. This research shows how much of a performance improvement the GPU can be at calculating finite state machines compared to the CPU.



**Fig. 3.** Relative speed up of GPU Performance [13]

In 2010, Richmond also created a cellular level agent-based simulation, once again using FLAME [14]. This simulation with FLAME allowed for simple implementation of FSMs using XML model files. These model files specify things such as agent memory variables, states and transitions functions that switch agents between states. There are also GPU schema extensions that specify things such as the max population size, discrete agents, continues agents and communication between agents. With these model files it removes the modeler away from having to fully having to understand the architecture thus allowing them to spend more time designing and developing the finite state machines themselves. This is becoming more and more important within games development. Game developers should be able to add and remove elements from the game with having to know as little code as possible because this speeds up overall application development.

Memory allocation is extreme important when dealing with GPGPU because there is not a large amount of shared data and the positions of the agents in memory can make a huge different. Agents can be sorted into similar state based groups, which reduces discrepancies between agents, allows specific look ups to be done at the same time and if some states are more intensive than others it timing may become a problem however, if they are grouped together you can generally estimate how long an agent in a group is going to take to be executed. At the end of the paper they also talk about how it could be improved further by increasing the detail of the cell models through the use of hierarchical modelling.

1. Conclusion

Through the years there have been massive developments in processing artificial intelligence on the graphics card, starting from using shaders to exploit the potential performance benefits of using the GPU. Even though this method isn’t ideal, it still succeeded in processing state based agents on the graphics card and shows there are unique ways of implementing FSMs other than lots of if statements. Now that there are languages made for exploiting the GPU, many researchers are attempting different methods of implementing AI onto the GPU and working out ways in which is can be improved.

Overall, all testing of the performance benefits have proved positive in almost every way, with massive magnitudes in some tests however, with these performance increases also comes constraints. There are a large number of constraints a developer will have to think about when it comes to implementing these systems on a GPU. These constraints range from power of the GPU to the actual brand of the graphics card. If these constraints aren’t though about early in development, it is going to heavily affect the range of players that are going to be able to play the game. If the developer can get past these constraints, it opens the door to potentially massive improvements to the game but also to the development of the game. Some solutions allow easier development of non-coding sections of the game so that designers can easily implement what they want or need without needing to know much coding or the architecture of the environment they are implementing on.

1. References

[1] Anderson, E. F. (2003). Playing Smart - Artificial Intelligence in Computer Games.

[2] Carneiro, T. (2001). A New Parallel Schema for Branch-and-Bound Algorithms Using GPGPU . 23rd International Symposium on Computer Architecture and High Performance Computing.

[3] Richmond, P. (2009). A High Performance Agent Based Modelling Framework. International Conference on Autonomous Agents and Multiagent Systems.

[4] D'Souza, R. M. (2007). SUGARSCAPE ON STEROIDS: SIMULATING OVER A MILLION AGENTS AT INTERACTIVE RATES. Proceedings of Agent2007 conference.

[5] Cuda Parallel Computing. (n.d.). Retrieved March 12, 2015, from Nvidia: <http://www.nvidia.co.uk/object/cuda-parallel-computing-uk.html>

[6] OpenCL. (n.d.). Retrieved March 12, 2015, from Khronos: <https://www.khronos.org/opencl/>

[7] Compute Shader. (n.d.). Retrieved March 12, 2015, from Microsoft: <https://msdn.microsoft.com/en-us/library/windows/desktop/ff476331%28v=vs.85%29.aspx>

[9] Thompson, T. (2008). An Evaluation of the Benefits of Look-Ahead in Pac-Man. IEEE Symposium on Computational Intelligence and Games.

[10 ] Nareyek, A. (2004). AI in Computer Games.

[11] Bleiweiss, A. (2008). GPU Accelerated Pathfinding. Graphics Hardware.

[12] Rudomín, I. (2005). Fragment Shaders for Agent Animation using Finite State Machines. Simulation Modelling Practice and Theory, 741-751.

[13] Joselli, M. (2009). GpuWars: Design and Implementation of a GPGPU. 09 VIII Brazilian Symposium on Digital Games and Entertain.

[14] Richmond, P. (2010). High performance cellular level agent-based simulation with FLAME for the GPU. Briefings in Bioinformatics, 334-347.