Applicability of GPGPU Computing to Real-Time AI Solutions in Games

William Blewitt, Gary Ushaw, and Graham Morgan

Abstract

This work reviews developments in General Purpose Computing on Graphics Processor Units (GPGPU Computing) from the perspective of video game related artificial intelligence. We present an overview of the field, beginning with early shader language solutions and continuing to discuss three accessible platforms for GPGPU development: CUDA, OpenCL and DirectCompute. Consideration is given to the commercial and practical realities which hinder the adoption of GPGPU solutions within video game AI and developments in GPGPU computing directly relevant to common AI practises within the video games industry are reviewed in depth.

Index Terms

Artificial Intelligence, Games, GPGPU, GPU Computing.

I. Introduction

Recent developments in GPU technology have, over the past four years in particular, inspired heightened research interest in the field of general purpose GPU computation. Areas of focus have varied from solvers for complex mathematics, physics and life sciences simulations to high performance computing and search optimisation [1, 2].

Historically, within the video games industry, hardware limitations and restrictions have compelled developers to optimise their code so as to make best feasible use of their eventual release platform. While occasionally the needs of the industry have prompted PC hardware conceptualisation and development [3], commercial concerns often stymie industrial adoption of that development as companies seek to maximise the potential market of their game. Adoption of such technologies has been more voluminous in cases where existing hardware that a potential customer can be assumed to possess already is utilised in novel and useful ways [4, 5].

In the context of artificial intelligence, attempts were made in 2006, by a company called AIseek [6], to pursue the same model that AGEIA initially adopted: the proposed development of a specialist co-processor card, this one performing tasks related to artificial intelligence rather than physics. Subsequent lack of industrial interest had several causes, not least of which related to the difficulties inherent in generating 'distinction' between game AI and other properties, and the lack of uniformity where artificial intelligence functions are concerned, but an undeniable

W. Blewitt, G. Ushaw & G. Morgan are with the School of Computing Science, Newcastle University, Newcastle, NE1 7RU, United Kingdom. Correspondence E-mail: w.f.blewitt@ncl.ac.uk

aspect was an understandable unwillingness to commit products for development solely for consumers willing to purchase a new, hitherto unnecessary piece of hardware.

The same commercial concerns, both from a context of player enjoyment and limited computational resources, have often driven a conceptual wedge between artificial intelligence research and artificial intelligence as it is applied in games [7]. The issue of ensuring an agent within a game environment is always a vehicle for player enjoyment, with artificial intelligence that reflects this [8], is a largely immobile constraint on the part of game developers. The ever-expanding capacities and capabilities of the GPU [9], however, may provide a hitherto largely-untapped facility to extend or expand computing cycles devoted to artificial intelligence processing in games. The issues faced by those proposing the idea of an AI accelerator card, in terms of requiring another piece of hardware to perform the task, no longer obtain when GPU cycles can be utilised for the same purpose.

We must be aware of the additional technical constraints that apply to currently available GPUs which impact their suitability for processing AI algorithms in a gaming context. Often multiple cores share relatively small cache resources; without careful management of this memory, naive implementations of algorithms on the GPU can lead to memory access bottlenecks. In addition, the utilisation of the GPU for general purpose computation often requires extensive copying operations both to and from the graphics memory of the system; this is specifically the case in situations where the CPU is expected to act upon the results produced by the GPU in some fashion, and the overhead involved is often non-trivial. These concerns, in conjunction with others discussed in depth later in this publication, conceptually limit the adoption of GPGPU computation in real-time game AI to tasks which can be processed within a frame of game time, although recent hardware advances go some way to mitigating these issues [10].

In this work, we explore the strides made in academic research using GPUs to accelerate tasks traditionally associated with artificial intelligence. We do so, however, from a perspective sympathetic to the real-world requirements of game developers, mindful that much artificial intelligence research is of limited utility in game development as a function of the aforementioned 'player enjoyment' constraint.

Initially, we shall discuss the currently available methods and languages by which the computing power of the GPU may be brought to bear on AI-related tasks. Later, we present some discourse regarding historical concerns in communicating relevance of artificial intelligence research to the video games industry, and conversely communicating the needs of that industry to the research establishment.

Subsequently, we consider published evaluations of the potential development platforms and their relative suitability from both a computational and industrial standpoint. Following this, we consider specifically the practises most often applied within the games industry and discuss in explicit terms how the developments in academic research can beneficially impact them. Our work then draws to a conclusion.

A. GPGPU Languages

While the advent of GPGPU computing can be considered a relatively recent development in computer science, the concept of using the graphics processing unit of a computer for purposes other than rendering is not as recent as this might imply [11]. Indeed, even the idea of using the GPU as an AI facilitator and accelerator is not an entirely modern concept [12, 13].

Several early platforms which empowered developers to explore general purpose computing on the GPU are still used in that fashion today. While the specific purposes of such platforms, including the OpenGL Shading Language (GLSL) [14], and NVIDIA's Cg and Microsoft's HLSL shader languages [15], were primarily to leverage hardware resources in a graphical context, they provided useful platforms for early GPGPU computing [16]. Indeed, such techniques are still relevant and currently employed in the context of game technology [17], and have been discussed explicitly in the context of artificial intelligence algorithms [18].

As the field has matured, building upon the groundwork laid by these legacy techniques, three modern GPGPU programming languages have received increased attention: NVIDIA's Compute Unified Device Architecture (CUDA) [19], the Khronos Group's Open Computing Language (OpenCL) [20], and Microsoft's DirectCompute [21].

1) CUDA: CUDA is a proprietary development language designed to give the programmer access to the computational capabilities of an NVIDIA GPU. Released in November 2006, the NVIDIA GeForce 8800GTX was the first consumer GPU capable of processing CUDA code. NVIDIA claimed that CUDA brought about a floating point performance increase, relative to shader code (such as GLSL), of a factor in excess of four when compared to performance figures from two years prior [22].

In the years since its release, CUDA has been the facilitator for significant academic research and development [23], as well as ongoing investigations into, and refinements of, its own versatility and optimisation [24].

2) OpenCL: Originally developed by Apple to be released in 2008 as part of their Mac OSX Snow Leopard [25], OpenCL is a programming framework for programs intended to be executed across heterogeneous platforms; in this context, heterogeneous meaning computers possessing CPUs, GPUs, FPGAs and other appropriate processors [26].

OpenCL, as an open standard, is not proprietarily limited to a single vendor's processors, granting it commercial flexibility from the perspective of platform-specific development. Like CUDA, OpenCL has been the subject of academic and industrial research both as a platform for development [27] and in terms of analysis of its own properties, performance and potential [28, 29].

3) DirectCompute: A more recent development than CUDA and OpenCL, DirectCompute forms part of the Microsoft DirectX 11 API and, by virtue of backwards compatibility, functions on both DirectX 11 and DirectX 10-compliant GPUs [30]. Its purpose is to provide GPGPU support for programmers using the DirectX API, which is of particular industrial relevance to game developers.

Like OpenCL, DirectCompute code functions on the graphics processors of multiple vendors, including NVIDIA GPUs; AMD GPUs and on-CPU graphics accelerators; and, very recently, Intel's HD4000 on-CPU graphics solution [31]. Unlike OpenCL and CUDA, DirectCompute is software-platform dependent, due to its relationship with the DirectX 11 API, and carries an intrinsic requirement that the target platform be running the Microsoft Windows Vista operating system or later.

B. Research AI from a Gaming Viewpoint

The divide between academic artificial intelligence research and artificial intelligence as it is generally applied and understood within the video games industry is often characterised as the 'traditional' disparity between scientists and engineers [7]. In specific terms, academic research into artificial intelligence often provides solutions to problems that the video games industry does not require answers to; moreover, academic research into artificial intelligence can often generate scientifically valuable conclusions that are nevertheless impractical for the video games industry to employ. Indeed, the techniques and approaches utilised by the academic and industrial establishments regarding the exploration of artificial intelligence are highly disparate, and only a small proportion of AI research is undertaken with broader industrial application in mind.

This is particularly relevant respective to the real-time artificial intelligence solutions which the video games industry necessarily employs, and particularly in the context of artificial intelligence solutions capable of providing a meaningful increase in perceived performance without impacting the cinematic framerates many video games are required to sustain. When discussing the adoption of GPGPU computing within commercial titles, this concern is compounded by concerns relating to the diversion of graphical resources away from rendering tasks. Artificial intelligence traditionally receives a smaller allocation of resources than graphics in general terms; actively diverting dedicated graphical resources to it would be a significant paradigm shift. Another concern in the context of GPU-based artificial intelligence is that often the performance benefits observed from GPU acceleration of a given algorithm are reached with large data sets or, in the context of per-agent simulations, quantities of agents that are rarely seen in games.

One often advocated 'solution' for the problem of mutual understanding between academic AI researchers and video game engineers, if it *is* a problem in the traditional sense of the word, is to attempt to facilitate the understanding within the video games industry that the products of academic research can have implications for them that are not apparent at first sight. Another, less common approach, is to directly target research interests and developments such that they are sympathetic to the existing needs of the video games industry and remaining mindful of the implications this has upon both purpose and platform.

In this article, we attempt to follow both approaches. Initially we consider traditional 'artificial intelligence' concerns that directly and regularly impact the video games industry, and consider the fashion in which research into GPU-based AI might support those needs. Subsequent to that, however, we discuss the products of academic research into artificial intelligence and consider how the video games industry might find relevance within them. We begin with some discussion of platform considerations, mindful of industrial concerns and constraints.

II. Comparison of GPGPU Development Platforms

As discussed previously, our interest in the field of GPGPU computing is primarily sympathetic to the commercial sensibilities of the video games industry. That being the case, we present in this section some detailed discussion of how the currently available, easily accessible GPGPU development platforms compare in terms of performance; specifically, we consider CUDA, OpenCL and DirectCompute.

Following that, we discuss the often-overlooked industrial concerns relating to the fashion in which GPGPU computing is employed, and the commercial consequences which might impact the selection of development platform.

A. Real-World Performance Figures

In 2008, Shuai Che et al. [32] published a work reviewing the performance of general purpose applications implemented for the GPU through CUDA. Their attention was devoted specifically to assessing the effectiveness of the CUDA GPGPU development platform across a selection of application types, offering some specific suggestions on how to improve their performance on the GPU. Considering structured and unstructured grid problems, combinatorial logic problems, dynamic programming applications and data mining algorithms, they conclude that the GPGPU applications obtained ”impressive” speedups, and comment on the ease with which applications could be developed in CUDA when compared to traditional shader-based approaches.

More recently, Gunasekaran [33] published an overview of GPGPU computing on CUDA, directing its attention to the structural differences between CPU computing and heterogeneous computing optimised to take advantage of GPU capabilities. In his work, he stresses the necessity to optimise algorithms to take advantage of parallel architectures in order to gain the maximum benefit from GPGPU development, and concludes that GPGPU computing is likely to continue to drive hardware innovation for the forseeable future.

In their 2010 work, Thibieroz and Cebenoyan [34] discussed the optimisation practices necessary to make best use of the DirectCompute heterogeneous computing platform. In particular, they emphasised the necessity to process enough work on the GPU to demonstrate a gain in efficiency, and the often overlooked requirement for GPUs to serve their rendering role in most situations.

Komatsu et al. [35] presented work in 2010 seeking to evaluate performance and portability of programs written using the OpenCL development platform. Using two benchmark programs, and three CUDA programs adapted to utilise the OpenCL standard through replacement of keywords and API functions, the work demonstrates that the OpenCL implementations outperform naive, CPU-based implementations, but are themselves outperformed by the native CUDA implementations. Komatsu et al. suggest that the lower OpenCL performance they observed are a combined function of compiler capabilities and the universality of OpenCL precluding platform-specific optimisations on a machine level.

Similar work performed by Karimi et al. [36] was undertaken to perform direct comparisons between applications built on the CUDA platform and equivalent applications utilising the OpenCL development platform. As previously, Karimi et al. observed superior kernel execution times in the CUDA implementations, despite the implementations running ”nearly identical code”. They conclude that in situations where achieving the highest possible performance is required, CUDA appears to be the better choice of development platform.

Harvey [37], in 2009, discussed his own experiences in porting code from the CUDA platform to OpenCL. In converting his implementation of a model for molecular dynamics from CUDA to OpenCL, he determined that it was feasible to maintain a single code base that worked for both platforms, and that OpenCL on NVIDIA hardware was almost as efficient as CUDA at the time of writing. He concluded that while CUDA was then the most mature GPGPU development platform available, OpenCL was a viable alternative.

In their 2011 work, Du et al. [38] published an extensive performance comparison between OpenCL and CUDA. Implementing a triangular solver and matrix multiplication solution on both software development platforms, they proceed to compare their performance across hardware platforms from both NVIDIA and AMD/ATI. In their conclusion, they highlight that overhead in OpenCL is large and should be minimised and, on NVIDIA hardware, OpenCL is ”fairly competitive” with CUDA.

Fang et al. [39] argue that perceived inefficiencies in the GPGPU applications developed on the OpenCL platform, relative to similar applications developed on the CUDA platform, are a function of ”unfair comparisons”. While they acknowledge that, for most applications, CUDA performs at best thirty percent more efficiently than OpenCL, they state that under a ”fair comparison”, OpenCL can achieve similar performance asserting that fairness is obtained through optimisations for OpenCL-compliant hardware. Their work focuses on developmental paradigms and flow, along with some particular consideration of the features CUDA is capable of supporting due to its hardware platform-specific nature that OpenCL does not in order to sustain its cross-platform portability.

The results of Fang et al. are partially supported by the conclusions of Fratarcangeli [40] who demonstrated in his 2011 work a practical and implemented cloth simulation comparing CUDA, OpenCL and GLSL with a traditional CPU implementation. In his conclusion, he states that there is evident superiority in the GPGPU implementation over the CPU implementation; he further points out that GLSL outperforms the more accessible GPGPU development platforms due to the nature of texture memory being both cached and optimised for two-dimensional local data. In terms of comparison between CUDA and OpenCL directly, however, he observes a uniformly lower performance in OpenCL which he attributes to difficulties in tuning of local and global work items.

Where there is a wealth of work directly comparing CUDA and OpenCL implementations of various GPU computing solutions, DirectCompute has received less academic attention. Zhang et al. [41] consider the use of DirectCompute in the context of computer vision. Their work revolves around background generation for object detection in traffic monitoring, and they present and test a DirectCompute-based solution. They observe a performance increase over and above a CPU-based solution of almost 70%.

Lastly, we consider published work contesting the generally accepted wisdom that GPGPU computation grants order of magnitude performance increases over optimised, multi-threaded traditional computing. In their 2010 work, Lee et al. [42] present a set of fourteen throughput computing kernels, with optimised implementations for both CPU and GPGPU platforms. In their conclusion, they argue that their results demonstrate that GPGPU implementations on their hardware setup averaged a speedup factor of 2.5.

Similarly, in their 2011 work, Gregg and Hazelwood [43] argue that when memory transfer times are included in GPU computational activities, GPGPU applications can demonstrate a slowdown of between two and fifty times when compared to a traditional CPU-based implementation. Benchmarking eleven applications, they conclude that much reporting regarding the efficiency of GPGPU computation overlooks the overhead generated when data is transferred between local and global memory, and propose a more revised and open taxonomy which identifies why such overhead is overlooked in the context of any algorithm that does not include it within benchmarking.

B. Game Development Practical Considerations

In the context of game development, particularly when considering development for home computers of potentially widely varying configurations, the exploration of GPGPU solutions to traditional programming tasks demands a level of critical attention that moves beyond mere floating-point throughput benefits.

The rendering capabilities of graphics processing hardware have increased by orders of magnitude in recent years, primarily driven by the joint requirements of higher definition displays and closer-to-photo-realistic rendering. In addition, the minimum GPU requirement to run any PC game remains the primary limiting factor in terms of that game's potential market.

Against that backdrop, it is unsurprising that the enthusiasm of the games industry to adopt GPGPU computing practises has been subdued in comparison to the academic response indicated by the wealth of research considered in this work. To increase the workload on the GPU carries, by logical extrapolation, either a limitation to the rendering capabilities of the targeted hardware, making the game arbitrarily less visually appealing, or a limitation of target sales market by increasing the specifications necessary to run the game.

As such, the value of GPGPU computation to the games industry is best emphasised through research which demonstrates that a consumer graphics card can sustain both the GPGPU enhancements to which the research obtains and a fully rendered environment running at a cinematic framerate, such as the works of Shopf et al. [44].

Quite aside from the graphical considerations are the arguments about which development platform to adopt were a company to implement, in this context, GPU-based artificial intelligence solutions. Unlike physics simulation, which the success of optional-activation proprietary engines such as PhysX have demonstrated can be conceptually scalable, the artificial intelligence of any game development cannot be treated in that fashion.

To put it another way a commercial title utilising PhysX, a physics calculation platform bound to contemporary NVIDIA graphics cards, can still be purchased and played by consumers using graphics cards from a different vendor, simply with less optimised physics-based visuals. Providing that sort of scaling in artificial intelligence requires far more complex decisions to be taken in early stages of development, if it is even feasible in the genre of a specific game to begin with. Ultimately, any non-transferable increase in artificial intelligence computation that impacts actual gameplay, rather than cosmetic enhancements to game experience, inherently changes the game from platform to platform.

As such, selection of a GPGPU platform can have profound impact upon the limitation of potential consumer base, or increase in development cost. We also should not overlook the current disparity in hardware between personal computer platforms and console platforms, which necessarily impacts the suitability of GPGPU techniques for titles being released across multiple platforms.

Of the three easily accessible GPGPU development platforms this work has considered, only one can be considered universally portable on both a hardware platform and software platform level, and that only in the context of personal computer hardware. OpenCL as a standard was envisioned with just that requirement in mind, and much work has been performed exploring its real-world portability [45].

A consequence of that portability, as has been indicated by the comparison studies published since the release of OpenCL and referenced earlier in this section, is that in real-world applications OpenCL often performs less optimally than NVIDIA's CUDA platform. But in selecting CUDA as her development platform of choice, a PC game developer immediately slashes her target market through the exclusion of customers with AMD/ATI graphics hardware and those using Intel graphics solutions.

One potential solution to this issue was presented by Jacob et al. [46] in 2010. Their tool, CUDACL, was designed to enable applications written for one platform to be swiftly and economically converted to run on the other. Taking into account perceived inefficiencies in OpenCL applications relative to their direct CUDA equivalents, this style of solution invites an approach similar to that taken with the PhysX engine, whereby GPU performance of AI calculations might be faster on the hardware of a specific vendor, but still function on the hardware of any vendor.

Another consideration is the limitation of software platform compatibility. The DirectCompute API, for example, limits the potential market of a given title to the Windows family of operating systems (Vista or later). While this is obviously of concern to developers wishing to publish titles which take advantage of the hardware universality found in modern computers using Mac OSX-based systems, it is in many ways a less profound concern than that of alienating graphics hardware vendors. Indeed, in situations where a game engine has been developed with the DirectX API in mind, such as the most recent iteration of the Unreal Engine, there is no inherent commercial consequence to adopting the DirectCompute platform, saving those already discussed in terms of consequence to graphics processing.

In addition, most AAA developers release titles across multiple platforms, including games consoles, simultaneously, and have mechanisms in place to handle parallel development across disparate hardware. With the advent of fixed-specification, high performance personal computer platforms, such as Valve's *Steam PC*, there may be commercial sustainability in developing with such fixed specifications in mind.

Further to this, developments by CPU vendors AMD and Intel in recent times have provided OpenCL and DirectCompute-compliant graphics accelerators as onboard additions to the CPU. While these solutions were, initially, primarily designed to permit energy-efficient gaming for low-specification applications, the latest generations have demonstrated suitability to manage the graphics in AAA titles at moderate resolutions. In most cases, however, game developers require that the consumer have a discrete graphics card in their system as an element of the minimum technical specifications of the product. As such, these on-board graphics solutions provide a potential avenue for the pursuit of 'AI-acceleration' through OpenCL or DirectCompute utilising processor scheduling that would otherwise be untapped.

In the context of interfaces designed to maximise versatility of a chosen GPGPU platform of development, JCUDA was presented by Yonghong Yan et al. [47] in 2009. Designed as a user-friendly interface to accelerate Java programs on CUDA-compliant GPUs, it was tested using four Java Grande benchmarks and demonstrated speedups ranging from six times to over one-hundred-and-twenty times. A similar concept proposed by Dotzler et al. [48] in 2010, extended the idea of CUDA-accelerated Java to the Open Multi-Processing (OpenMP) standard, again highlighting the value of platform portability. Both of these contributions highlight useful versatility in development platform, inviting game developers operating in Java to leverage GPU resources for general purpose computation.

III. GPU Developments in Existing Game AI Practises

In this section, we present a review of advances GPGPU computing has provided in areas of artificial intelligence that the video games industry already readily employs.

Zamith et al. [49] discussed in explicit terms how the advent of GPGPU computing could directly affect architectural concepts in game engine development. This work is of interest primarily due to the approach taken with respect to acknowleding the primacy of heterogeneous hardware architectures and working within the constraints inherent to them so as to optimise hardware utilisation, rather than promoting a complete architectural shift.

For our part, however, we consider specific aspects of ”artificial intelligence” as practised by the video games industry, broken down into four areas. First, we shall consider pathfinding and navigation as implemented on the GPU. After that, we review developments in GPGPU research with an impact on line of sight and neighbour detection. Third, we consider fashions in which the GPU has impacted traditional search systems. Lastly, we review implementations of finite state machines on GPGPU platforms.

A. Pathfinding and Navigation on the GPU

Almost all video games require some form of navigational artificial intelligence to inform the behaviour of NPCs. This might be as simple as reciprocal movement between two points in two-dimensional space, as demonstrated in many platform adventure games of the 1980s, or as complex as a police car pursuing a player driven vehicle in a three-dimensional environment replete with other vehicles, destructible terrain, pedestrians, and any of a number of other hazards.

When discussing video game pathfinding, however, it is common to refer back to one of the most often and readily implemented algorithms for determining the shortest unobstructed path between two points, A* [50, 51]. Optimisations to A* are commonplace in game development [52, 53], as are refinements to its implementation to make it more versatile or intelligent [54]. At its heart, however, A* remains a cornerstone of game development and has been used in real-time path planning for decades.

Avi Bleiweiss [55] presented his CUDA-based GPGPU solution for A* pathfinding in 2008, alongside a similar implementation of the classical Dijkstra search algorithm. He addresses issues associated with implementing an irregular and divergent A* kernel on a hardware platform which optimally excels in regular, highly arithmetic processes; further to this, he presents an optimal solution to the issue of parallelising A*. His conclusions indicate a speedup of an order of magnitude over a two-threaded C++ A* algorithm running on a dual core CPU with Streaming SIMD Extensions.

Silva et al. [56] presented an evolution of Bleiweiss's work with their own implementation of A* for the GPU using CUDA. Usefully, their work explored the impact of larger maps than Bleiweiss's 340 nodes, managing to sustain a map of 2025 nodes with 65536 agents, or 300000 agents on a map of 400 nodes. They acknowledge in their conclusion that their relative speedup over the CPU implementation is lower than that obtained by Bleiweiss in 2008, and they observe that a limiting constraint remains the memory available on GPUs at the time of publication.

Somewhat earlier, in 2006, Zioma [57] proposed a pathfinding solution that offloaded the majority of the preprocessing workload onto the GPU. In his work, he addressed shader-based implementations of both the Dijkstra algorithm and the Floyd-Warshall algorithm, outlining the process of managing data structures as textures through the DirectX API. His results demonstrated modest speedups in the context of the GPGPU implementation of Dijkstra over the CPU implementation, less than a factor of six at optimal conditions; more usefully, the GPGPU implementation of the Floyd-Warshall algorithm demonstrated speedups relative to the CPU implementation of Dijkstra, in some cases, of over one hundred times.

These results highlight a point raised by Buluç et al. [58] in their 2010 work regarding the solution of path problems on the GPU. Utilising the CUDA platform, they implemented a pathfinding algorithm based upon block-recursive elimination. They stress in their conclusion that selecting an algorithm which makes best possible use of the hardware resources available to it is essential in designing any high performance computing solution, including those applied to GPUs; their results indicate that their optimised implementation outperformed a port of the ”most popular” algorithm to GPU.

An alternative navigation system proposed by Dapper et al. [59] and based upon an extension of Laplace's Equation, was implemented using the CUDA platform by Fischer et al. [60] in 2009. Generating a family of potential field functions lacking local minima, resolved as a boundary value problem, Fischer et al. tested their GPGPU implementation across three different map sizes. Their results demonstrated that the GPGPU implementation performed generally faster than the CPU implementation, with larger maps obtaining greater speedups; the optimal setup for parallelisation demonstrated a speedup of 56 times relative to the CPU implementation.

In 2008, Shopf et al. [44] demonstrated a highly successful implementation of multi-agent navigation on both a global and local level through GPGPU technology. Global navigation was managed through variable potential fields and a variant of Dijkstra's method known as the Fast Marching Method [61], guiding the agents along their general path. Noting that this solution for the eikonal equation became prohibitively expensive if applied to local navigation, Shopf et al. implemented a local avoidance model to avoid collisions against fine-grained obstacles. Their testing system setup was, by current standards, readily obtainable in a consumer computer, and demonstrated the ability to manage and render approximately 65,000 agents with a framerate in excess of 30 frames per second.

Bleiweiss [62] presented, in 2009, an approach to manage navigation using Velocity Obstacles [63] built on the CUDA platform and optimised for multi-agent systems. In the work, Bleiweiss addresses the issue of idling processors for managing scenarios with a low agent count through proposal and formulation of a velocity level, nested parallel solution. Bleiweiss observes a performance speedup of over four times, relative to the CPU implementation, and an interactive framerate of 18 frames per second during an evacuation scenario featuring 10,000 agents.

You Zhou and Ying Tan [64] published an implementation of a GPU-based particle swarm navigation solution built upon the CUDA platform, which they extended in 2011 [65] to manage swarms with multiple objectives. In the former case, they highlight the benefits of mass-parallelisation of processed work packets with respect to management of swarms, before presenting a parallelised solution to the computation of fitness values, updating particle state, and resolving velocity and position. They conclude that the running time of the GPGPU implementation is significantly shortened, relative to the running time of the CPU implementation, with certain situations demonstrating a speedup in excess of eleven times. Regarding their subsequent work on multi-objective swarms, they conclude that similar speedups are observed, though the GPGPU implementation is best applied to large-scale simulations.

In 2008, Passos et al. [66] presented a CUDA-platform implementation of flocking boids taking advantage of data structures optimised for applicability to mass-parallelised GPGPU computing. Providing an exhaustive description of the manner in which the data structures handling boid data are processed, they go on to demonstrate significant performance increases on the GPGPU implementation of boid navigation, relative to the CPU implementation, in situations where the number of boids exceeds 10000. Further to this, utilising consumer PC hardware, their GPGPU simulation processes 1048576 boids at a framerate of 25 frames per second.

Taking a different approach, focusing upon estimated self-occlusion, da Silva et al. [67] presented an implementation of flocking boids utilising GPU hardware using texture mapping and shader programming. Their experimental results, obtained using a less capable GPU than those of Passos et al. demonstrated a speedup of the GPGPU implementation relative to the CPU implementation for all values of boids, though the speedup was somewhat irregular at boid counts below 10000.

In 2012, dos Santos et al. [68] published a comprehensive work addressing the parallelisation of the FIPA architecture for multi-agent systems. Their test case explores the performance of their system when applied to A* pathfinding for a massively multi-agent environment. They conclude that their results call for future exploration into agent-oriented paradigms, implemented using the GPU.

Demeulemeester et al. [69] presented work in 2011 which also addressed the issue of path planning for large numbers of agents using the GPU. They utilised a hybridised approach which included an initial, coarse A* stage to guide an implementation of the continuum crowds algorithm in order to reduce its computational expense. Their results demonstrated successful, real-time rendering and path-planning for up to 100,000 agents using a common consumer graphics card.

B. Line of Sight and Detection on the GPU

Determination of detection of a target, and viable line of sight between an agent and a target or a player character and a target, are key artificial intelligence-related tasks in many game genres. They are of particular relevance to developers working on FPS-style games, and the optimisation of those tasks in any three-dimensional rendered environment can greatly reduce their processor scheduling requirements [70].

Manocha [71] highlighted the benefits of applying GPGPU computation to geometric algorithms, such as line of sight determination, in 2005. He notes ”more than an order-of-magnitude” improvements over contemporary CPU-based solutions to problems within that family, including collision detection. This point is re-emphasised by

Kalyan [72] who, in 2006, referred to line of sight calculations for battlefield simulations as one possible application for GPGPU technology.

Verdesca et al. [73] presented work in 2005 discussing the manner in which GPGPU computing could increase the processing speed of the One Semi-Automated Forces (OneSAF) Computer Generated Forces (CGF) simulation. OneSAF is a distributed computer simulation designed to simulate unit behaviour in military scenarios; importantly, it operates at entity-level, with agents in the simulation representing individual personnel, facilitating highly detailed behavioural projections.

One aspect of this work addressed the implementation of a GPGPU-based solution for line-of-sight computation, prompted by the impact that line-of-sight determination for thousands of agents within the simulation had upon runtime performance. They concluded from their benchmark performance that outsourcing line-of-sight calculations to the GPU provided a realtime increase in performance of twenty times.

The specific algorithm Verdesca et al. applied to OneSAF was first published by Salomon et al. [74] in 2004. Their experimental results demonstrated an average time per query of four microseconds, a constant time speedup in line of sight computation; they further highlight the fact that in many simulations, line-of-sight computation can be the most expensive within the simulation, and their belief that exploiting the GPU to that purpose is a valuable effort.

In his 2010 work, Britton [75] presents a CUDA-based implementation of recursive ray-tracing on the GPU. Within this implementation, he discusses ray-based determination of line-of-sight between a camera and an object as a means of determining which objects the renderer can ”see”, implemented on the GPU as a part of the overall renderer architecture.

In the context of agent detection, Jia Pan et al. [76] presented an efficient solution for *k*-nearest-neighbour computation on the GPU in 2010. Structured as a support for agent motion planning, their algorithm was parallelised and implemented on the CUDA development platform. Their experimental results indicated that their GPGPU implementation outperformed the comparative CPU implementation by a factor between twenty and forty, dependent upon the system’s constraints.

Peinado’s [77] work in 2010 addressed similar concerns, presenting an optimised solution for nearest-neighbour classification using the OpenCL development platform. His experimental results reinforced earlier points regarding the requirement to optimise GPU code to offset hardware limitations; in particular, he notes the manner in which memory access can have significant influence over performance.

In addition, the works of Passos et al. and de Silva et al., previously discussed in the context of path planning, have particular relevance to the issue of line of sight and neighbour detection utilising the GPU. Both implementations rely upon near neighbour detection; in the case of de Silva et al., self-occlusion is a fundamental aspect of their algorithm. In 2009, de Silva et al. [78] released an extension of their work presenting a more detailed consideration of agent vision. Their results reinforced earlier conclusions regarding the performance benefits of GPU utilisation and the value of self-occlusion, or visibility culling, to performance with minimal impact upon model behaviour.

The work of Passos et al. was later expanded upon by Joselli et al. [79] in their extension of the neighbourhood data structure into the neighbourhood grid and its implementation on the GPU as a means of governing the behaviour of large numbers of boids. Their results demonstrated a noticeable speed-up relative to traditional solutions through subdivision of environment, with a smaller memory footprint.

C. Search Algorithms on the GPU

Search algorithms play a significant role in any user-interactive software development, and are particularly relevant in games engineering. The most common application for search algorithms has, of course, already been touched upon in the context of pathfinding, as nodal navigation is an easily translateable concept [80]. An example of relevance to game development was published by Isla [81] in 2006, addressing the concept of target tracking and search through application of occupancy maps. In addition, techniques such as goal-oriented action planning [82] can be enhanced by accelerated search techniques.

Krüger et al. [83] published work in 2010 addressing the subject of a generic, hybrid local search algorithm developed on the CUDA platform for the GPU. Their specific implementation combined an evolutionary algorithm with a local search algorithm, the goal being to minimise search time overall. The observed speedup in performance, relative to a CPU implementation of the same process, varied between forty-seven times and ninety-five times, with the higher speedup values being observed at higher population sizes and iterations.

In their 2009 work, Edelkamp and Sulewski [84] explored the concept of parallel state space searching using the GPU. They outline the applicability of the search schema to obtaining solutions for games such as the Rubik's Cube and Sliding-Tile Puzzle, before presenting the implementation of a breadth-first heuristic search on the GPGPU platform. The experimental results indicated a speedup of thirty times, or more, for the GPGPU implementation compared with a CPU implementation tasked to the same search.

In 2010, Ryan [85] presented a CUDA-based implementation of the Boyer-Moore string searching algorithm, adapted for the GPGPU platform. Directing particular focus to the nature of GPU memory management, and the benefits or otherwise of ”mapping” and ”pinning” memory within a CUDA application, Ryan's work demonstrated that the GPU could accelerate the Boyer-Moore search algorithm by some sixty percent, but that mapping and pinning the memory did not increase the application's performance.

Rocki and Suda [86] considered the issue of parallel minimax tree searching on the GPU in 2009, implementing their solution using the CUDA platform. They highlight the relevance of the minimax algorithm to search of game trees, their particular implementation being based upon the rule set for the game Othello. Highlighting the fact that CUDA did not permit recursion at that time, requiring them to adopt a modified, iterative variant upon the minimax algorithm, they go on to present the fashion in which the problem is parallelised. Their experimental results led to their concluding that the GPGPU implementation only outperformed the CPU implementation when the algorithm was adapted to suit batch processing; indeed, they state that a direct port occasionally showed worse performance. In situations where a high level of parallelism existed, however, the GPGPU implementation showed a speedup of approximately thirty-two times, relative to the CPU implementation.

Rocki and Suda [87] extended this work in 2012, through exploration of the acceleration of Monte Carlo Tree

Search methods through GPGPU implementation. Again using Othello as the basis for their game scenario, they concluded that the GPGPU implementation outperformed the CPU implementation, with the GPU performing calculations equivalent to between sixty-four and one-hundred-and-twenty-eight CPUs.

Along similar lines, Bleiweiss [88] presented work in 2010 addressing the applicability of GPGPU computing to the field of zero-sum games, including three-dimensional naughts and crosses, Connect-4, and Othello (Reversi). Applying a variant of the minimax algorithm to a four-by-four-by-four naughts and crosses, Bleiweiss's results indicate that the GPGPU implementation is outperformed by the CPU implementation until four-thousand matches are played simultaneously, whereupon the parallelised nature of the GPU demonstrates noticeable speedup; at 16000 parallel matches, the speedup of the GPGPU implementation is over five times.

D. Finite State Machines on the GPU

Finite state machines (FSMs) are a mainstay of artificial intelligence within video games, and have been for some years. One of the most commonly referenced examples is that of Pac-Man, the ”Ghosts” in which were governed by finite state machines [89]. Optimisation and design of finite state machines to facilitate straightforward game development has been of some research interest historically [90].

In the context of finite automata implemented on the GPU, some significant efforts have been made even prior to the advent of easily accessible GPGPU development platforms such as OpenCL and CUDA. In 2005, Rudomín et al. [91] presented an implementation of finite state automata using fragment shaders through GLSL. While the work was primarily proof of concept, rather than performed with a view to comparing efficiency with traditional implementations, it successfully demonstrated the viability of implementing a predator-prey FSM on the GPU.

Hayenga and Shaw [92] explored the concept of extended finite automata implemented on the GPU and the Cell Broadband Engine using the RapidMind Development Platform. An extension of deterministic finite automata designed to circumvent their ”state explosion weakness”, extended finite automata are thus more versatile from a game development perspective. While Hayenga and Shaw's implementation found the RapidMind platform unsuited to their purposes, a purely GPGPU-based solution presented by Goyal et al. [93] demonstrated a speedup of up to nine times relative to a comparable CPU implementation.

In their 2009 paper, Joselli and Clua [94] implemented virtually an entire game engine on the GPU, featuring agents governed by finite state machines. They presented their CUDA-based solutions to governing agent behaviour, game physics and rendering, proposing in their conclusion that future work should include the implementation of fuzzy logic, artificial neural networks, and hierarchical state machines.

Leung et al. [95] presented a solver for real time collision detection in their 2010 work, referencing implementations on both the GPU and the Field Programmable Gate Array (FPGA). Describing their system as a finite state machine with thirty-eight states, Leung et al. discuss and compare the performance of both the GPU and FPGA implementations; they conclude that the FPGA outperforms the GPGPU implementation in general terms by a factor of eleven, though their experimental results indicate that the execution time of the GPGPU implementation is lower than that of the FPGA implementation at vertex counts in excess of two-hundred-and-fifty.

In terms of cellular automata, Richmond et al. [96] address the concept of cellular level agent-based simulations implemented through a GPGPU development platform known as the Flexible Large-Scale Agent Modelling Environment (FLAME). In their work they present a schema by which FSMs and extensions thereof can be implemented through their platform using extensible mark-up language (XML) model files. Their conclusions focus upon the successful implementation of their development platform, and its universality.

A final consideration in the context of finite state machines implemented on the GPU is inspired by the work of White et al. [97] in 2007. In their work, they propose the application of an SQL-based scripting language utilising queries to replace traditional FSM structure. Through this architecture, they demonstrate that their system can update the state of more than 12000 units, ten times per second, while a naive FSM implementation utilising the same hardware cannot scale to 1100 units. They conclude that their system is robust and expressive, faciliating both enhanced performance and greater versatility than traditional approaches.

Bakkum and Skadron [98] proposed, in their 2010 work, a method by which SQL database operations might be accelerated through usage of GPGPU technology, constructed on the CUDA platform. Their work focuses on accelerating SELECT queries, the queries which comprise the bulk of the SQL-based FSM proposed by White et al.; furthermore, Bakkum and Skadron devote significant attention to utilizing the memory hierarchy of the CUDA platform, with the databases upon which queries are acting being stored in global memory as they would ideally be in a game development scenario where on-GPU memory is preferably dedicated to graphical data. Their results demonstrate a speedup over a CPU-based implementation of the same queries of up to eighty times in scenarios where results transfer time is ignored, and up to thirty-five times in scenarios where it is not.

In his 2010 work, Hristov [99] proposed a similar acceleraton schema for SQL-like queries using the OpenCL development platform rather than CUDA. One consequence of this selection of development platform was that the resulting application could run on either the GPU or the CPU and while Hristov found the performance of the GPU to be limited by the low memory throughput between host and device for small amounts of data, the OpenCL implementation did show improvement over the traditional implementation when run on the CPU itself.

This, combined with the considerations of Bakkum and Skadron, gives significant encouragement to the proposition that the work of White et al. would generate significantly greater improvements in multi-agent behavioural governance if revisited with a view to optimising the SQL-style queries through a heterogeneous computing platform.

IV. Conclusion

This work has considered extensively fashions in which existing practises in video game development, which might traditionally fall outside the purvue of artificial intelligence research, have benefited from research into GPGPU computation. Such considerations have included references to observed, real-world speed benefits, and suggestions have been made regarding how current and prior work might be expanded in light of developments in GPGPU technology. Specifically, the work has proposed an exploration of the concepts suggested by White et al. [97] with respect to replacing traditional FSM architectures with SQL-style database queries, in light of the results obtained by Bakkum and Skadron [98] in their exploration of CUDA-accelerated SQL queries.

The work has presented discourse regarding commercial implications of adopting GPGPU techniques for the processing of artificial intelligence solutions. It has suggested paradigms that future GPGPU research undertaken with a view to appealing to that industry might wish to pursue, particularly regarding the design of prototype implementation and platform selection for portability.

Further, it has suggested fashions in which the commercial concerns of GPGPU development might be offset through utilisation of existing solutions, such as the CUDACL solution produced by Jacob et al. [46]. Similarly, the work has proposed further exploration of the manner in which the computational capabilities of existing hardware might be suitably leveraged in the context of video game AI. Specifically, the authors intend to investigate the heterogeneous computational capabilities of the on-CPU, OpenCL and DirectCompute-compliant graphical solutions developed in recent times by Intel and AMD/ATI, and their viability as artificial intelligence coprocessors.

The work has concluded that in order for GPU-based artificial intelligence solutions to truly appeal to the video games industry, they must be pursued in a fashion sensitive to the commercial realities in which that industry is compelled to operate.

ACKNOWLEDGEMENTS

The authors wish to thank the review panel, and Phil Scott of NVIDIA, for their valuable contributions in support of this work.

REFERENCES

[1] M. Abshoff and C. Pernet, “Efficient exact linear algebra over gpu,” Presentation, August 2008.

[2] T. Carneiro, A. E. Muritiba, M. Negreiros, and G. A. L. de Campos, “A new parallel schema for branch-and-bound algorithms using gpgpu,” in *Proceedings of the 23rd International Symposium on Computer Architecture and High Performance Computing*, 2011.

[3] AGEIA, “White paper: Ageia physx,” *Computer Power User*, vol. 5, no. 10, pp. 44–47, October 2005.

[4] M. Mittring and B. Dudash, “The technology behind the directx 11 unreal engine ”samaritan” demo,” in *Proceedings of Game Developers Conference 2011*, 2011.

[5] J. Craighead, J. Burke, and R. Murphy, “Using the unity game engine to develop sarge: A case study,” in *Proceedings of the 2008 Simulation Workshop at the International Conference on Intelligent Robots and Systems*, 2008.

[6] AIseek, “Aiseek - intelligence for new worlds,” White Paper, 2006.

[7] C. Baekkelund, “Academic ai research and relations with the game industry,” in *AI Game Programming Wisdom 3*, S. Rabin, Ed. Charles River Media, Inc., Massachusetts, 2006, ch. 1.7, pp. 77–88.

[8] L. Lidén, “Artificial stupidity: The art of intentional mistakes,” in *AI Game Programming Wisdom 2*, S. Rabin, Ed. Charles River Media, Inc., Massachusetts, 2004, ch. 1.4, pp. 41–48.

[9] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, “Nvidia tesla: A unified graphics and computing architecture,” *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[10] NVIDIA, “Nvidia’s next generation cuda compute architecture: Kepler gk110,” White Paper, 2012.

[11] M. Macedonia, “The gpu enters computing’s mainstream,” *Computer*, vol. 36, no. 10, pp. 106–108, 2003.

[12] U. Erra, R. D. Chiara, V. Scarano, and M. Tatafiore, “Massive simulation using gpu of a distributed behavioral model of a flock with obstacle avoidance,” in *Proceedings of the International Workshop on Vision, Modeling and Visualization*, 2004.

[13] C. J. Darken, E. R. Pursel, and J. S. Correia, “Ai on the gpu,” in *ACM Workshop on General Purpose Computing on Graphics Processors*, 2004.

[14] J. Kessenich, D. Baldwin, and R. Rost, *The OpenGL Shading Language, Version 1.1*, OpenGL Architecture Review Board, 2004.

[15] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," in *Eurographics 2005, State of the Art Reports*, August 2005, pp. 21–51.

[16] L. Latta, "Building a million particle system," in *Proceedings of Game Developers Conference 2004*, 2004.

[17] D. J. Morris, E. F. Anderson, and C. Peters, "A modular framework for deformation and fracture using gpu shaders," in *Proceedings of VSMM 2012: 18th International Conference on Virtual Systems and Multimedia*, 2012.

[18] A. Seoane and A. Jaspe, "Ia algorithm acceleration using gpus." in *Encyclopedia of Artificial Intelligence*, J. R. Rabual, J. Dorado, and A. Pazos, Eds. IGI Global, 2009, pp. 873–878.

[19] NVIDIA, "Cuda libraries," Presentation, 2007.

[20] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in Science & Engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[21] J. Yang and L. Howes, "Advanced directx technology: Directcompute by example," in *Proceedings of Game Developers Conference Europe 2010*, 2010.

[22] D. Kirk, "Nvidia cuda software and gpu parallel computing architecture," in *Proceedings of the 6th International Symposium on Memory Management*, 2007.

[23] J. C. Thibault and I. Senocak, "Cuda implementation of a navier-stokes solver on multi-gpu desktop platforms for incompressible flows," in *Proceedings of the 47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*, 2009.

[24] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W. mei W. Hwu, "Optimization principles and application performance evaluation of a multithreaded gpu using cuda," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2008.

[25] Apple, "Apple previews mac os x snow leopard to developers," Press Release, June 2008.

[26] P. Jääskeläinen, C. S. de La Lama, P. Huerta, and J. Takala, "Opencl-based design methodology for application-specific processors." in *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, F. J. Kurdahi and J. Takala, Eds. IEEE, 2010, pp. 223–230.

[27] S. de Smet, "An opencl fast fourier transformation," Project Report, 2011.

[28] R. Ferrer, J. Planas, P. Bellens, A. Duran, M. Gonzalez, X. Martorell, R. M. Badia, E. Ayguade, and J. Labarta, "Optimizing the exploitation of multicore processors and gpus with openmp and opencl," in *Proceedings of the 23rd International Conference on Languages and Compilers for Parallel Computing*, 2010, pp. 215–229.

[29] E. Chung, P. Milder, J. Hoe, and K. Mai, "Single-chip heterogeneous computing: Does the future include custom logic, fpgas, and gpgpus?" in *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, dec. 2010, pp. 225 –236.

[30] T. Ni, "Direct compute - bring gpu computing to the mainstream," in *GPU Technology Conference*, 2009.

[31] W. Engel, "Microsoft directcompute on intel ivy bridge processor graphics," Intel Corporation/Confetti Inc., Tech. Rep., 2012.

[32] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using cuda," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370–1380, October 2008.

[33] G. Gunasekaran, "Study of performance for the cpu and gpu architecture," *International Journal of Research in IT, Management and Engineering*, vol. 2, no. 2, pp. 33–44, February 2012.

[34] N. Thibieroz and C. Cebenoyan, "Directcompute performance on dx11 hardware," in *Proceedings of Game Developers Conference 2010*, 2010.

[35] K. Komatsu, K. Sato, Y. Arai, K. Koyama, H. Takizawa, and H. Kobayashi, "Evaluating performance and portability of opencl programs," in *Proceedings of the Fifth International Workshop on Automatic Performance Tuning*, 2010.

[36] K. Karimi, N. G. Dickson, and F. Hamze, "A performance comparison of cuda and opencl," *CoRR*, vol. abs/1005.2581, 2010.

[37] M. Harvey, "Experiences porting from cuda to opencl," Presentation, December 2009.

[38] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra, "From cuda to opencl: Towards a performance-portable solution for multi-platform gpu programming," 2011.

[39] J. Fang, A. L. Varbanescu, and H. Sips, "A comprehensive performance comparison of cuda and opencl," in *Proceedings of ICPP 2011: International Conference on Parallel Processing*, 2011, pp. 216–225.

[40] M. Fratarcangeli, "Gpgpu cloth simulation using glsl, opencl, and cuda," in *Game Engine Gems 2*, E. Lengyel, Ed. A K Peters / CRC Press, 2011, ch. 22, pp. 365–378.

[41] X. Zhang, B. Wang, and C. Geng, “Gpu-based background generation method,” in *Proceedings of the IET International Communication Conference on Wireless Mobile and Computing (CCWMC 2011)*, 2011.

[42] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, “Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu,” in *Proceedings of the 37th International Symposium on Computer Architecture*, June 2010, pp. 451–460.

[43] C. Gregg and K. Hazelwood, “Where is the data? why you cannot debate cpu vs. gpu performance without the answer,” in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, ser. ISPASS ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 134–144. [Online]. Available: http://dx.doi.org/10.1109/ISPASS.2011.5762730

[44] J. Shopf, J. Barczak, C. Oat, and N. Tatarchuk, “March of the froblins: Simulation and rendering massive crowds of intelligent and detailed creatures on gpu,” in *SIGGRAPH*. ACM, August 2008, pp. 52–101.

[45] S. Rul, H. Vandierendonck, J. D’Haene, and K. De Bosschere, “An experimental study on performance portability of opencl kernels,” in *Proceedings of the 2010 Symposium on Application Accelerators in High Performance Computing*, 2010.

[46] F. Jacob, D. Whittaker, S. Thapaliya, P. Bangalore, M. Mernik, and J. Gray, “Cudacl: A tool for cuda and opencl programmers,” in *Proceedings of the 2010 International Conference on High Performance Computing (HiPC)*, 2010.

[47] Y. Yan, M. Grossman, and V. Sarkar, “Jcuda: A programmer-friendly interface for accelerating java programs with cuda,” in *Euro-Par*. Springer-Verlag Berlin Heidelberg, 2009, pp. 887–899.

[48] G. Dotzler, R. Veldema, and M. Klemm, “Jcudamp: Openmp/java on cuda,” in *Proceedings of the 3rd International Workshop on Multicore Software Engineering*, 2010.

[49] M. Zamith, E. Clua, A. Conci, and A. Montenegro, “Parallel processing between gpu and cpu: Concepts in a game architecture,” in *Computer Graphics, Imaging and Visualisation, 2007. CGIV ’07*, aug. 2007, pp. 115 –120.

[50] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions of Systems Science and Cybernetics*, vol. 4, pp. 100–107, 1968.

[51] D. Higgins, “Generic a* pathfinding,” in *AI Game Programming Wisdom*, S. Rabin, Ed. Charles River Media, Inc., Massachusetts, 2002, ch. 3.2, pp. 114–121.

[52] ——, “Pathfinding design architecture,” in *AI Game Programming Wisdom*, S. Rabin, Ed. Charles River Media, Inc., Massachusetts, 2002, ch. 3.3, pp. 122–132.

[53] B. Anguelov, “Video game pathfinding and improvements to discrete search on grid-based maps,” Master’s thesis, Faculty of Engineering, Built Environment and Information Technology, University of Pretoria, June 2011.

[54] D. Silver, “Cooperative pathfinding,” in *AI Game Programming Wisdom 3*, S. Rabin, Ed. Charles River Media, Inc., Massachusetts, 2006, ch. 2.1, pp. 99–112.

[55] A. Bleiweiss, “Gpu accelerated pathfinding,” in *Graphics Hardware*, D. Luebke and J. D. Owens, Eds. The Eurographics Association, 2008.

[56] A. Silva, F. Rocha, A. Santos, G. Ramalho, and V. Teichreib, “Gpu pathfinding optimization,” in *Proceedings of the 2011 Brazilian Symposium on Games and Digital Entertainment (SBGames’11)*, 2011.

[57] R. Zioma, “Preprocessed pathfinding using the gpu,” in *AI Game Programming Wisdom 3*, S. Rabin, Ed. Charles River Media, Inc., Massachusetts, 2006, ch. 2.4, pp. 141–156.

[58] A. Buluç, J. R. Gilbert, and C. Budak, “Solving path problems on the gpu,” *Parallel Comput.*, vol. 36, no. 5-6, pp. 241–253, Jun. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2009.12.002

[59] F. Dapper, E. Prestes, and L. P. Nedel, “Generating steering behaviors for virtual humanoids using bvp control,” in *Proceedings of CGI*, 2007.

[60] L. G. Fischer, R. Silveira, and L. Nedel, “Gpu accelerated path-planning for multi-agents in virtual environments,” in *Proceedings of the 8th Brazilian Symposium on Games and Digital Entertainment (SBGames’09)*, 2009, pp. 101–110.

[61] J. N. Tsitsiklis, “Efficient algorithms for globally optimal trajectories,” *IEEE Transactions on Automatic Control*, vol. 40, pp. 1528–1538, 1995.

[62] A. Bleiweiss, “Multi agent navigation on the gpu,” in *Proceedings of Game Developers Conference 2009*, 2009.

[63] P. Fiorini and Z. Shiller, “Motion planning in dynamic environments using velocity obstacles,” *The International Journal of Robotics Research*, vol. 17, pp. 760–772, 1998.

[64] Y. Zhou and Y. Tan, "Gpu-based parallel particle swarm optimization," in *Proceedings of the 2009 IEEE Congress on Evolutionary Computation (IEEE CEC 2009)*, 2009, pp. 1493–1500.

[65] ——, "Gpu-based parallel multi-objective particle swarm optimization," *International Journal of Artificial Intelligence*, vol. 7, pp. 125–141, 2011.

[66] E. Passos, M. Joselli, M. Zamith, J. Rocha, A. Montenegro, E. Clua, A. Conci, and B. Feijó, "Supermassive crowd simulation on gpu based on emergent behavior," in *Proceedings of the 7th Brazilian Symposium on Games and Digital Entertainment (SBGames'08)*, 2008, pp. 81–86.

[67] A. R. da Silva, W. S. Lages, and L. Chaimowicz, "Improving boids algorithm in gpu using estimated self occlusion," in *Proceedings of the 7th Brazilian Symposium on Games and Digital Entertainment (SBGames'08)*, November 2008, pp. 41–46.

[68] L. G. O. dos Santos, E. W. G. Clua, and F. C. Bernardini, "A parallel fipa architecture based on gpu for games and real time simulations," in *Proceedings of the 11th International Conference on Entertainment Computing (ICEC 2012)*, 2012.

[69] A. Demeulemeester, C.-F. Hollemeersch, P. Mees, B. Pieters, P. Lambert, and R. V. de Walle, "Hybrid path planning for massive crowd simulation on the gpu," in *MIG'11 Proceedings of the 4th international conference on Motion in Games*, 2011.

[70] T. Vykruta, "Simple and efficient line-of-sight for 3d landscapes," in *AI Game Programming Wisdom*, S. Rabin, Ed. Charles River Media, Inc., Massachusetts, 2002, ch. 2.7, pp. 83–89.

[71] D. Manocha, "General-purpose computations using graphics processors," *Computer*, vol. 38, pp. 85–88, 2005.

[72] K. S. Perumalla, "Discrete-event execution alternatives on general purpose graphical processing units (gpgpus)," in *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, 2006, pp. 74–81.

[73] M. Verdesca, J. Munro, and M. Hoffman, "Using graphics processor units to accelerate onesaf: A case study in technology transition," in *Proceedings of the 2005 Interservice/Industry Training, Simulation and Education Conference (I/ITSEC 2005)*, 2005.

[74] B. Salomon, N. Govindaraju, A. Sud, R. Gayle, M. Lin, D. Manocha, B. Butler, M. Bauer, A. Rodriquez, L. Eifert, A. Rubel, and M. Macedonia, "Accelerating line of sight computation using graphics processing units," in *Proceedings of the 24th Army Science Conference*, 2004.

[75] A. D. Britton, "Full cuda implementation of gpgpu recursive ray-tracing," Master's thesis, Department of Computer Graphics Technology, Purdue University, 2010.

[76] J. Pan, C. Lauterbach, and D. Manocha, "Efficient nearest-neighbor computation for gpu-based motion planning," in *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 2243–2248.

[77] I. J. Peinado, "Exploiting contemporary architectures for fast nearest neighbor classification," Master's thesis, Department of Computer Architecture, Universitat Politècnica de Catalunya, January 2010.

[78] A. R. da Silva, W. S. Lages, and L. Chaimowicz, "Boids that see: Using self-occlusion for simulating large groups on gpus," *ACM Computers in Entertainment*, vol. 7, pp. 51:1–51:20, 2009.

[79] M. Joselli, E. B. Passos, M. Zamith, E. Clua, A. Montenegro, and B. Feijó, "A neighbourhood grid data structure for massive 3d crowd simulation on gpu," in *Proceedings of the 8th Brazilian Symposium on Games and Digital Entertainment (SBGames'09)*, 2009.

[80] P. Tozour, "Search space representations," in *AI Game Programming Wisdom 2*, S. Rabin, Ed. Charles River Media, Inc., Massachusetts, 2004, ch. 2.1, pp. 85–102.

[81] D. Isla, "Probabilistic target tracking and search using occupancy maps," in *AI Game Programming Wisdom 3*, S. Rabin, Ed. Charles River Media, Inc., Massachusetts, 2006, ch. 5.1, pp. 379–388.

[82] J. Orkin, "Applying goal-oriented action planning to games," in *AI Game Programming Wisdom 2*, S. Rabin, Ed. Charles River Media, Inc., Massachusetts, 2004, ch. 3.4, pp. 217–228.

[83] F. Krüger, O. Maitre, S. Jiménez, L. Baumes, and P. Collet, "Speedups between x70 and x120 for a generic local search (memetic) algorithm on a single gpgpu chip," *Lecture Notes in Computer Science*, vol. 6024, pp. 501–511, 2010.

[84] S. Edelkamp and D. Sulewski, "Parallel state space search on the gpu," in *Proceedings of the 2009 International Symposium on Combinatorial Search (SoCS)*, 2009.

[85] K. Ryan, "An evaluation of gpus for accelerating a selected string searching algorithm," VeriSign Labs, Tech. Rep., 2010.

[86] K. Rocki and R. Suda, "Parallel minimax tree searching on gpu," in *Proceedings of the 8th International Conference on Parallel Processing and Applied Mathematics*, 2009, pp. 449–456.

[87] ——, "Accelerating parallel monte carlo tree search using cuda," in *GPU Technology Conference*, 2012.

[88] A. Bleiweiss, “Playing zero-sum games on the gpu,” in *GPU Technology Conference 2010*, 2010.

[89] T. Thompson, L. McMillan, J. Levine, and A. Andrew, “An evaluaton of the benefits of look-ahead in pac-man,” in *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games. CIG’08*, 2008, pp. 310–315.

[90] E. Yiskis, “Finite-state machine scripting language for designers,” in *AI Game Programming Wisdom 2*, S. Rabin, Ed. Charles River Media, Inc., Massachusetts, 2004, ch. 5.4, pp. 319–326.

[91] I. Rudomín, E. Millán, and B. Hernández, “Fragment shaders for agent animation using finite state machines,” *Simulation Modelling Practice and Theory*, vol. 13, pp. 741–751, 2005.

[92] M. Hayenga and M. Shaw, “Extended finite automata on stream-oriented architectures using rapidmind,” 2007.

[93] N. Goyal, J. Ormont, R. Smith, K. Sankaralingam, and C. Estan, “Signature matching in network processing using simd/gpu architectures,” The University of Wisconsin-Madison, Department of Computer Sciences, Tech. Rep., 2008.

[94] M. Joselli and E. Clua, “Gpu wars: Design and implementation of a gpgpu game,” in *Proceedings of the 8th Brazilian Symposium on Games and Digital Entertainment (SBGames’09)*, 2009, pp. 132–140.

[95] B. Leung, C.-H. Wu, S. Memik, and S. Mehrotra, “An interior point optimization solver for real time inter-frame collision detection: Exploring resource-accuracy-platform tradeoffs,” in *Proceedings of the 2010 International Conference on Field Programmable Logic and Applications (FPL)*, 31 2010-sept. 2 2010, pp. 113 –118.

[96] P. Richmond, D. Walker, S. Coakley, and D. Romano, “High performance cellular level agent-based simulation with flame for the gpu,” Briefing in Bioinformatics, 2010.

[97] W. White, A. Demers, C. Koch, J. Gehrke, and R. Rajagopalan, “Scaling games to epic proportions,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2007, pp. 31–42.

[98] P. Bakkum and K. Skadron, “Accelerating sql database operations on a gpu with cuda,” in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, 2010.

[99] V. Hristov, “Performance evaluation of query processing algorithms on gpgpus,” Master’s thesis, School of Informatics, University of Edinburgh, 2010.

William Blewitt obtained a B.Sc degree in Physics with Space Science and Technology at the University of Leicester in 2004, before earning a M.Sc degree in Computational Intelligence and Robotics at De Montfort University in 2006. During his PhD in Computer Science at De Montfort University, he researched computationally inexpensive emotion modelling for AI agents before studying a M.Sc in Computer Game Engineering at Newcastle University. He joined the game technology research group at Newcastle University in 2012, where his research interests include modelling believable agents in real-time systems, heterogeneous computing solutions for AI, and optimisations of general purpose GPU computing for commercial application.

Gary Ushaw received the B.Sc degree in Electronics from the University of Manchester Institute of Science and Technology (UMIST) in 1987, and the PhD in Signal Processing from the University of Edinburgh in 1995. From 1987 to 1991 he worked as an electrical engineer on large scale public projects for communications and control, working with CEGB, British Rail, and the combined electricity generating boards of India. From 1995 to 2011, he worked in the games industry as software engineer and later as engineering manager, focusing on high-end console gaming with publishers including Ubisoft, Sony, Rockstar, BBC and Atari. In 2011 he joined the teaching staff at Newcastle University School of Computing Science, concentrating on video game engineering, rehabilitative gaming, and multicore systems.

Graham Morgan gained his PhD in 1999 and spent the time since studying a variety of areas in computing. With a research background in systems, Graham has published many articles on the engineering challenges related to video game development. This has included collision detection, online gaming, physics engines, graphics, AI and multi-core exploitation techniques.