Hybrid architecture for AI-based RTS games

Antonio Maciá-Lillo*, Antonio Jimeno-Morenilla†, Higinio Mora‡, Eduard Duta§
Department of Computer Science and Technology, University of Alicante
Email: *a.macia@ua.es, †jimeno@ua.es, ‡hmora@ua.es, §aed6@gcloud.ua.es

***Abstract*—Video games have evolved into a key part of modern culture and a major economic force, with the global market projected to reach $282.30 billion in 2024. As technology advances, video games increasingly demand high computing power, often requiring specialized hardware for optimal performance. Real-time strategy games, in particular, are computationally intensive, with complex artificial intelligence algorithms that simulate numerous units and behaviors in real-time. Specialized gaming PCs are used to run these games, which have a dedicated GPU. Due to the usefulness of GPUs besides gaming, modern processors usually include an integrated GPU, specially in the laptop market. We propose a hybrid architecture that utilizes both the dedicated GPU and the integrated GPU simultaneously, to accelerate AI and physics simulations in video games. The hybrid approach aims to maximize the utilization of all available resources. The AI and physics computations are offloaded from the dedicated GPU to the integrated GPU. Therefore, the dedicated GPU can be used exclusively for rendering, resulting in improved performance. We implemented this architecture in a custom-built game engine using OpenGL for graphics rendering and OpenCL for general-purpose GPU computations. Experimental results highlight the performance characteristics of the hybrid architecture, including the challenges of working with the two devices and multi-tenant GPU interference.**

***Index Terms*—iGPU, dGPU, Architecture, Game Engine, AI based games.**

I. Introduction

VIDEO games are a widespread media of entertainment nowadays. They have become an intrinsic part of modern culture. According to Statista [1], the total market revenue of the video game industry is projected to be 282.30 billion US dollars in 2024, and it is expected to keep growing. Video games are a product of the computing technology of their time. As technology keeps advancing rapidly, so does video games. In fact, video games are usually a high performance task that needs specialized hardware to run [2]. However, they have to take into account the software and hardware limitations of the environment, specially if they intend to reach a broad audience of heterogeneous devices [3].

Artificial Intelligence (AI) has been important in video games since the very beginning [4], although some genres make more extensive use than others. Real-Time Strategy Games (RTS) games are a video game genre focused on strategy and resource planning, where the players continuously make decisions and perform actions, and the goal usually is to destroy the enemy player [5]. Generally, in video games, rendering graphics is a demanding task, specially when the game uses 3D graphics [3]. On top of that, in RTS games, AI is specially challenging. In these games, there are hundreds of units that have to be rendered and their actions simulated. Furthermore, the behaviors have to be computed within real-time constraints. The result is complex AI algorithms that require high amount of computing resources to work.

Video games usually require a lot of computing power, which means they need systems with powerful devices to run [6]. For PC games, the commercial concept of gaming PC has been developed for high performance computers. A gaming PC is a performance oriented computer with the best commodity hardware available [7]. The high hardware requirements of video games leaves out more modest user with less capable devices. Video games need to balance the game characteristics in order to keep the hardware requirements down to be able to reach a wider audience. It is of utmost importance to squeeze the highest performance possible from the device the game is running, making use of all the components and resources available.

The search for better performance ended up introducing specific computing devices for video games: the Graphics Processing Unit (GPU). Its original task was to accelerate the rendering of computer graphics with its highly parallel nature, but it is used also to perform other tasks like AI and Machine Learning (ML) [8], [9], [10], [11], [12], [13], High Performance Computing (HPC) [8], physics simulations [14], [15], [16], [17], [18] or Big Data analytics [8], in what is known as General-Purpose GPU computations (GPGPU). As the usefulness of the GPU device started to be seen in fields outside video games, processor manufacturers started including it into the CPU chip. The embedded version of the GPU, known as integrated GPU (iGPU), is usually less powerful than the external, dedicated GPU (dGPU), which is connected to the motherboard by a system bus. In gaming computer setups, the dGPU is commonly used, whereas the integrated GPU is often included with most processors. However, some processor models come without an iGPU. When both cards are present, the iGPU is underused. Games are generally prepared to use only one graphic card, and when they are able to use multiple cards, it is in case the user has more than one dGPU, which does not account for the concrete characteristics of the iGPU device [19], [20].

Our proposal is the use of a hybrid architecture which uses the dGPU and the iGPU simultaneously, with the aim of increasing the performance of video games with expensive AI and physics computations, by fully utilizing the resources available. The novelty of this research is that the game engine uses the dGPU for rendering, whereas, simultaneously, the iGPU is used to perform GPGPU computations to accelerate the game systems. A custom-built game engine has been used, using the OpenGL library, in order to test the architecture. The GPGPU computations use the OpenCL library. We extensively

test the hybrid architecture and discuss the performance results and characteristics of such architecture, including the specifics of iGPU devices and multi-tenant GPU performance interference.

The paper is structured as follows: Beside the introduction, a review of current literature is performed. After that, the details of the architecture and game engine are explained. Next, the experiments are presented, where the performance of the hybrid architecture is tested. Lastly, the conclusions of this study are shown, and the limitations and future work of the study are commented.

II. Review

A. iGPU presence in game systems

Integrated Graphics Processing Units offer built-in graphics capabilities directly within the CPU [21]. Unlike dedicated GPUs, which are standalone graphics cards with specialized memory, iGPUs share system resources, making them a cost-effective and energy-efficient solution for handling basic to moderate graphical tasks. In modern processors, they are integrated into the CPU die, sharing the Last Level Cache (LLC) with the other processor cores [22]. They are widely used in laptops, desktops, and mobile devices, providing sufficient performance for everyday applications such as web browsing, media playback, and office productivity. Advances in iGPU technology have also improved their capabilities, enabling some to support tasks that previously required dedicated GPUs, including light gaming, video editing, and rendering [22].

The presence of iGPUs in modern computers is widespread, including gaming oriented PCs. Intel desktop processors from the latest 14th and 13th generation have a version with iGPU and a version without it, corresponding with the CPU models with the letter F. The iGPU models in those processors are Intel UHD 7xx. On the other hand, all Intel laptop oriented processors of the 14th generation include an intel UHD 7xx or Iris iGPU. In the case of AMD, all desktop and laptop processors include integrated graphics. The desktop processors of the 9000 generation include a Radeon 6xxM iGPU. The laptop processors of the 7000 series include a Radeon 6xxM iGPU. The laptop 8000 series includes a Radeon 7xxM or a Ryzen Z1 iGPU. Lastly, the laptop AI 300 series includes a Radeon 8xxM iGPU.

B. AI and video games

The aim of AI in video games is to enhance gameplay, improve player engagement, and streamline game development processes [23]. Uses of AI include the implementation of Non Player Characters (NPCs) to simulate intelligence and entertain the player [24]. It has a powerful impact on gameplay experience. Games modify AI parameters to alter the fuzzy logic behavior and modify game difficulty [25]. A well-designed AI can improve the entertainment value of the game [26].

AI in video games is constrained to the limited computational resources present on the systems the game is projected to run. In contrast to other fields like graphics, AI is not easily scalable for systems with different resources. Better or worse AI affects the gameplay experience, so the lower end systems that are supported by the game usually define the game AI performance [27].

Among games, AI is crucial in strategy games. The AI has to be applied at the level of strategic opponents and also to individual units [28]. Among strategy games, RTS games have the AIs with most demanding resources.

RTS games are a genre of video games which require managing different kind of units and resources in real-time. Players have to position and maneuver units and structures under their control to secure areas of the map and/or destroy the assets of their opponents [29]. Age of Empires, Starcraft, Warcraft and Total War series are examples of games of the RTS genre.

As personal computers emerged, the RTS game genre grew in popularity [30]. These type of video games are considered simplified military simulations, where the player indirectly manages units and buildings, issuing orders from an overhead view. The objective is usually to gather resources, build an infrastructure and an army, and eliminate enemy players. It differs from turn-based strategy games, where players take turns and have ample time to plan their moves [31], in that players must continuously make decisions and take actions [5]. A typical match of this kind of games involves players starting with a few units and buildings and limited resources, but have nearby resources available to be gathered. The player is expected to use these resources to expand its infrastructure, army, and research technology to be able to attack other players. This conflict is usually driven by the depletion of resources near the base, which forces players to expand and fight for the control of further sources. Depending on the game, a player is out when it looses all buildings, fails a specific objective, or surrenders.

An RTS game is usually played by more than two players, that can be human or AI. AI in RTS games is complex as it has to mimic human behavior in the game [29]. Furthermore, in contrast with turn based strategy, where specific time between turns can be reserved for the AI computations, the game simulation in RTS games is performed in real-time, usually in a fixed constant frame rate. The AI of these games is constrained by the lack of CPU resources [5], [30], [32]. Therefore, improving the performance of AI algorithms in RTS games is of vital importance.

Advance in commercial AI algorithms is stagnant. AI in RTS games generally uses scripted behavior, with manually predetermined rules and behaviors [33]. The main technique used when developing an RTS game AI is behavior trees [34]. There are several reasons that hold back the implementation of more innovative techniques. It is perceived too risky to apply current research on AI in production systems, specially in the case of nondeterministic techniques, which are difficult to understand, test or debug. Commercial game development is time-constrained, so the extra time needed to develop new and complex techniques is perceived as a major limitation. Finally, more complex techniques have a higher run-time and resource usage, which would reduce the resources available to other elements of the game such as graphics [33] .

RTS games traditionally have been of academic interest in the field of academic AI, as these games offer several AI research problems that have applications outside the game domain [5]. Here, the term AI refers to machine learning algorithms that are trying to play the games as competitive as humans. Buro & Furtak [32] provide a list of the main challenges that an AI faces in this type of games. These are adversarial real-time planning, decision-making under uncertainty, opponent modelling, spatial and temporal reasoning, resource management, player collaboration, and path-finding. Churchill et al. [35] perform fast heuristic searches using Alpha-Beta search to decide moves in localized combat scenarios. Advances in AI technology have also been brought to this field. Andersen et al. [36] present Deep RTS, an RTS game engine, to then implement a reinforcement learning algorithm for the AI. Wender & Watson [37] introduce a navigation component based on a hybrid case-based reasoning. The model combines case-based reasoning with reinforcement learning to achieve the goals of damage avoidance and target approximation during movement. Lucero et al. [38] use principles of Human-Autonomy Teaming (HAT) and eXplainable AI (XAI) to create an AI agent that provides recommendations, assistance in tasks, and teaches the player to play better. The underlying technology used is deep learning, including fully connected and Convolutional Neural Networks (CNN). Machalewski et al. [39] perform semi-automatic construction of AI agents adapting and tuning behavior trees by generalizing given human gameplay. Lorthioir & Inoue [40] present an adaptive Bayesian AI model to inference the most probable build order used by a player at any given time. Liu et al. [41] propose a deep reinforcement learning model for playing card-based RTS games, and implements it to play the game Clash Royale.

Recently, the interest on developing AI algorithms able to play these games is on the rise due to the involvement of big IT companies [42]. Google's DeepMind group developed AlphaStar, which uses deep neural networks, including CNN and Long Short-Term Memory, with the A3C reinforcement learning algorithm to play Starcraft 2 [43]. It first succeeded in winning a professional player in December 2018 [44]. OpenAI group created OpenAI Five, an AI trained with deep reinforcement learning that was able to win the world champion team in the game Dota 2 in April 2019 [45].

However, improving the resources available for AI will benefit games besides RTS. With the rise of modern machine learning approaches such as deep learning, and due to an increase in GPU power and capabilities, there is an increase of interest in the literature to integrate deep learning approaches into video game systems [46]. From improved versions of traditional algorithms with neural networks to entirely new deep learning approaches.

C. GPU architectures in games

GPUs are devices designed originally to render graphics from video games or animations. However, these devices have shown its usefulness in GPGPU computations. Specific frameworks and libraries have been created to this end. There are platforms specialized only in GPGPU computations. From these, the most popular is CUDA [47]. It was introduced by Nvidia, and it only works with their GPUs. Similarly, AMD has ROCm [48] as its way to use their GPUs for GPGPU applications. OpenCL [49] is an industry standard to do GPGPU computations. It works on Nvidia, AMD and Intel devices. To improve support of GPGPU computations in games and other graphical applications, graphical libraries such as OpenGL, Vulkan and Direct3D started supporting it natively. This technique is known as compute shader. A compute shader is a shader specialized for computing that runs on its own computing pipeline, separated from the graphics pipeline [50].

There are several games that have used GPUs in a computing architecture to perform other tasks besides rendering. The first tasks that games implemented on GPUs were physics operations, as even specific computing devices such as PhysX PPU were released, but at the end, using the GPU was seen as more convenient [51]. Examples of these early physics operations on the GPU include collision detection [14], [15], cloth simulation [16], dynamic terrain simulation [17], and fire simulation [18]. Today, physics computing in the GPU is well established, as it is used in main game physics engines such as Havok FX [52], PhysX [9] or Bullet [53].

As AI has become more complex and uses computing intensive algorithms, there is recent academic interest in executing AI algorithms in these devices. Stéphane & Éric [9] use the GPU to execute a binary AI planning algorithms that computes plans in order to control the behaviors of thousands of NPCs. Mahale et al. [10] accelerate the game tree search algorithm using the GPU. McMillan et al. [11] implement popular game path-finding algorithms on the GPU, including a novel parallel version of the collaborative diffusion algorithm. The A* algorithm, which is used in path-finding, has also been implemented using the GPU by Zhou & Zeng [12]. Shopf et al. [13] perform a continuum based path finding method on the GPU to simulate the movement of a massive crowd of NPCs. It uses a cost function that it is optimized using a parallel algorithm called Fast Iterative Method.

Other research games have tried to use all the computing resources available to obtain maximum performance. Nah et al. [54] use the GPU with the CPU to perform a ray tracing algorithm to render the frames. In the same line, Tayyub & Khan [55] use a CPU-GPU architecture to perform collision detection. Joselli et al. [56] present an adaptive game loop that is able to dynamically allocate tasks in different cores of multiple CPUs and GPUs, choosing the best device for the task by evaluating performance in each device over time. When the amount of resources is limited, Cloud Gaming increases the capabilities of the devices by using a computing outsourcing architecture. Computations are sent to the cloud to be performed by devices with more resources. The main way it operates is by sending the user input to the cloud, computing there the game frames, which are sent back to the user for visualization [57]. However, there are other alternatives. Sanchez et al. [58] propose a mobile cloud computing architecture where an Operating System (OS) scheduler is used that treats the Cloud as another available computing unit of the platform.

D. Hybrid GPU architecture

A hybrid architecture in the context of High Performance Computing (HPC) is the use of a computing architecture with different computing resources with the intent of maximizing the performance by maximizing the utilization of the computing resources [59].

The use of GPUs with CPUs in a hybrid architecture has been studied extensively by the literature. Sharma et al. [59] present a CPU-GPU approach for Monte Carlo simulations that maximizes the utilization of CPUs and GPUs in modern workstations. A load balancer is used to dynamically allocate specific operations to the GPU and the CPU, and a significant speed-up factor is achieved. Zidan et al. [60] present a technique to improve database applications. Using a hybrid CPU-GPU platform, they solve inefficiencies on the Smith-Waterman algorithm for determining similarities between nucleotide or protein sequences when computing the similarity matrix with a GPU algorithm, as in short queries result in GPU threads being idle. To avoid this, short queries are computed in the CPU while long queries in the GPU. Papadrakakis et al. [61] use the same type of architecture to accelerate a domain decomposition method. The different operations that the method uses, such as the Cholesky direct solver, dot product, or matrix-vector multiplications, are implemented specifically for this architecture. Zhu et al. [62] use a high performance CPU-GPU hybrid system to train node embeddings. Different tasks are executed in these two types of devices that adapt to its characteristics. This system is able to combine a CPU with several GPUs. Chen et al. [63] use the same hybrid architecture to perform the Fast Fourier Transform. The work is divided in simpler primitives that are computed using the GPU and the CPU at the same time. Tomov et al. [64] present a hybrid LU factorization algorithm that utilizes the CPU and the GPU. In this case, specific tasks of the operation are performed in both devices in parallel, with synchronization steps between these two devices.

However, the focus of this study is on hybrid architectures that utilize the integrated Graphics Processing Unit (iGPU). There are several studies that have shown the benefits of using the iGPU in a hybrid architecture combined with the main processor, as it is present in the almost all consumer grade processors. Lupescu et al. [65] improve sorting algorithm performance using the iGPU in a pipeline where it performs partial sorts of slices of the initial data set, and the newly partially sorted set is passed to the CPU sorting algorithm. Kuhrt et al. [66] use the iGPU to accelerate the operation of pattern matching in SQL databases. In this case, the use of an iGPU has the advantage of having shared memory with the CPU, which eliminates the latency of data copies that happen with a dGPU. Campos et al. [67] present a computing framework that utilizes the CPU cores and the iGPU to perform epistasis detection. Lupescu & Ţăpuş [68] test a hybrid architecture with the CPU and the iGPU with good results that performs the hash-table insert operation. Tseng et al. [69] make use of a virtual switch hybrid architecture. It leverages computing between the CPU and the iGPU in order to improve the throughput.

Lastly, some studies use the iGPU in conjunction with other dedicated GPUs, to fully utilize all the resources available in the system. Karnagel et al. [70] implement a database system that can efficiently exploit highly heterogeneous hardware environments. It can leverage between the different compute units of the system, like dGPUs, iGPU and the CPU, to efficiently process queries concurrently. Lupescu et al. [71] use different hybrid architectures that combine the CPU, dGPU and the iGPU to perform AES encryption. With the hardware at the time of the article, as processors include AES hardware acceleration, with the AES-NI instructions, using only the CPU gives the best performance to cost ratio, although using a dGPU in conjunction with the CPU gives the best performance. Combining the CPU with the iGPU provides a small performance boost over using only the CPU. Using the three devices (dGPU, iGPU and CPU) obtains worse performance compared to using only the CPU and the dGPU. Performance degrades, specially on the CPU, when more devices are used. Peek et al. [72] propose a hybrid architecture for real-time graphic applications such as video games and Virtual Reality (VR), where the number of frames is augmented using an image warping algorithm run in the iGPU, while the graphics engine is using the dedicated GPU to generate the base frames for the algorithm. With this method, the perceived frame rate and latency are reduced.

E. Findings

Overall, there is interest in running GPGPU computations in games. The GPU is already being extensively used to accelerate physics computations. Also, as AI technology is improving, accelerating AI algorithms on the GPU has the possibility to offer more complex and immersive AI in games, specially in RTS games. However, as rendering a game is a computing intensive task, the little computing power left in the CPU and the GPU to run these algorithms is very restrictive, which limits the scope of what the games are able to do. With these restraints, it is of interest of game developers to make their game engines utilize all the resources available in the system. The iGPU is a hardware that is present in almost all modern processors. However, it has received very little attention. From the review of the literature, we have only found one article [72] that utilized the iGPU in conjunction with the other computing devices for real-time graphic applications to obtain better performance. In contrast, our proposal is to utilize the iGPU in a hybrid computing architecture to accelerate other tasks like physics or AI besides rendering, which can help games with great amounts of physics and AI calculations.

III. Proposed hybrid architecture

The main proposal of this work is a hybrid architecture for video games and other real-time video systems. This architecture leverages all available computing devices in typical gaming computer systems. Specifically, the architecture aims to make use of the integrated graphics card, in the scenario where both a dGPU and an iGPU are present. Figure 1 shows how the architecture uses all the devices of the system.

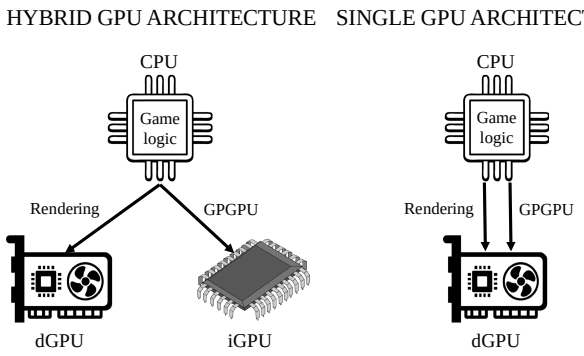


Fig. 1: Resource usage in the traditional, single GPU architecture and in the proposed hybrid GPU architecture

Games, specially the RTS genre, make use of game logic systems that have an extremely parallel nature, and benefit from using a GPU device, as our review confirms. Typically, those GPGPU computations would be run on the dGPU. However, in the proposed hybrid architecture, they are run on the iGPU, increasing the resource usage of the system.

The main advantage of using the integrated card is that it offloads the dGPU of the GPGPU computation. This is critical, as in video games, the dGPU is already busy with the rendering tasks. As a result, both rendering and GPGPU computations compete for the use of the dGPU, which impacts the overall performance of the system. The hybrid architecture enables the use of GPU acceleration in game logic systems without increasing the load on the dGPU, by taking advantage of the already present iGPU device.

IV. Test system

A. Game engine

For this work, a custom game engine has been developed. This decision was made in order to fully tune and customize the engine. It is true that using a real game endows the hybrid model with more empirical strength, however, it means not being able to control each of the rendering stages for the experiments. A video game prototype has been designed using the custom engine. Although it is not commercial, it has all the essential elements to model the characteristics of RTS games. To evaluate the effectiveness of the hybrid architecture, the prototype allows moving the AI and Physics calculations freely between the CPU, dGPU and iGPU.

The novelty of the engine is that it is able to use all computing resources of the system (CPU, dGPU and iGPU) at the same time to improve performance. This section describes the design characteristics of the used engine, which were chosen to be similar to what commercial game engines typically use.

The Entity Component System [73], [74], [75] (ECS) design pattern has been used. The entities represent any game object, the components are game data, and the systems operate with the data of the components. Figure 2 shows the class diagram of the engine implemented. Characteristics of the chosen design are:

- Components are associated to only one entity
- Components are stored in vectors of components of the same type, so they are aligned in memory
- Systems mainly use data from one component to operate, although they can use multiple components
- Data dependencies are addressed through shared components between systems

Concrete functionality is therefore encapsulated in the different systems, that operate with concrete components.

The render system uses the OpenGL library to draw the frames. It features 3D graphics with texture mapping and dynamic lighting from multiple sources to create a significant load on the GPU.

To accelerate the game systems, the OpenCL library is used. OpenCL is a standard to create parallel applications. OpenCL applications are developed in a programming language similar to C99. OpenCL provides an abstraction on the architecture. Computations are divided in work items. Every work item represents a thread, and each thread has its own private memory. Work items are organized into work groups, that have its own shared memory called local memory. Lastly, there is a global device memory, which every work group can access, where data has to be copied from the main memory (RAM). This abstraction fits well with current GPU architectures. Also, OpenCL is multi-platform, allowing to execute the same code in GPUs and CPUs from different vendors, as device manufacturers are in charge of the implementation of the standard in their devices. This has been useful in the experiments, where changing code execution between different GPUs has been needed.

B. Game prototype

To conduct the experiments, a game prototype has been created, that models the characteristics of RTS games. In that sense, it has extensive AI and Collision computations to simulate the behavior of big numbers of NPCs.

In the game, a predefined number of entities are organized into two battalions, and an AI algorithm calculates the position of every unit. Collisions are calculated between entities. Entity logic can be computed without drawing the entity model. Therefore, it is possible to tune the amount of load for render, collision and AI tasks.

Figure 3 shows a screen capture of the prototype. An already made 3D model has been used for the soldiers. It has a CC-BY-3 license. Its original authors are Konstantin Maystrenko (Concept), AnthonyMyers (Modeller) and Danimal (Retopo/Rigging/Animation) [76].

C. Collisions

The collision system checks if two entities overlap. To this end, the bounding box collision algorithm is used [77]. The algorithm is used extensively in modern game engines as a way to provide fast yet effective collision detection [78]. It resolves collisions by simplifying complex 3D objects into simple 3D boxes. A collision happens when the bounding boxes of two objects overlap in all axes. Figure 4 shows how it works in a two-dimensional environment.

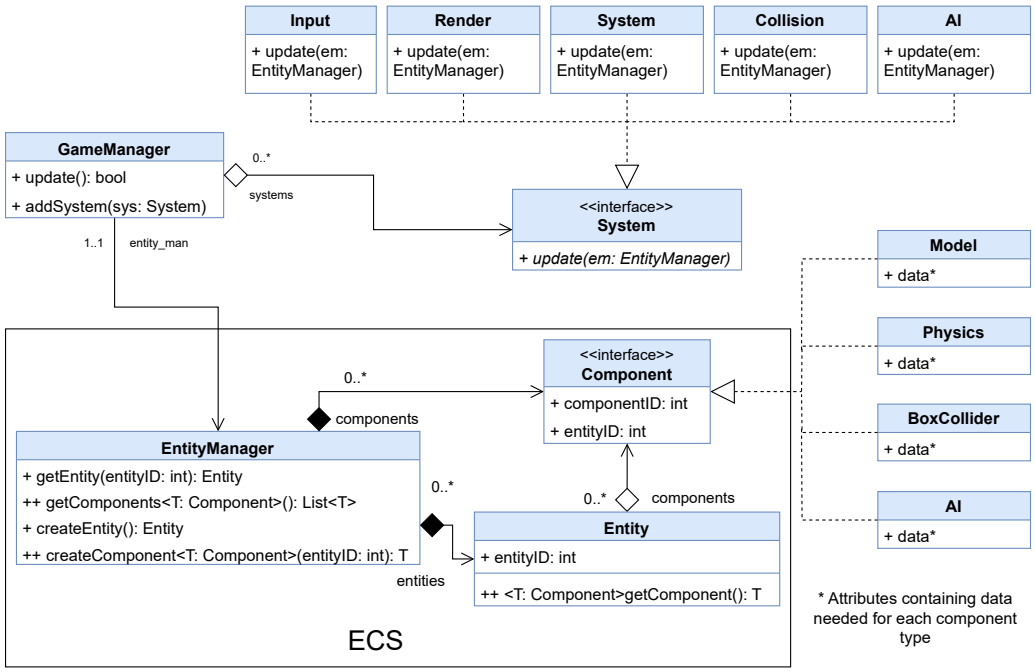
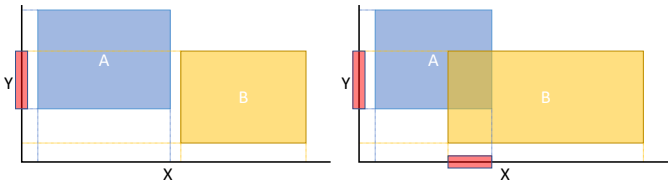


Fig. 2: Class diagram of the game engine

Fig. 3: Screen capture of the game prototype



(a) A and B do not collide (b) A and B collide

Fig. 4: Example of bounding box collision detection in a 2D environment

The basic approach to this algorithm involves checking all other entities for each entity. It has a temporal complexity of $O(n^2)$, where two nested loops go through all the entities. A minor optimization has been used to avoid checking pairs of entities twice. It involves starting the second loop on the index of the first loop. Although there are further optimizations to the algorithm that reduce its complexity, such as axis aligned collision [79], or the use of data structures like quadtrees [80], they were discarded to simplify the implementation.

The GPU version of the algorithm uses a thread for every pair of entities. Inside a thread, the operations to check if the pair of entities collide is performed. This means that for n entities, an amount in the range of n^2 threads is launched, exploiting the massively parallel nature of the GPU. The amounts of threads per block used is 64, to maximize GPU utilization.

D. AI algorithm

The AI for the game prototype has been implemented using the Jaya algorithm [81], [82]. This is an optimization method, that aims to get an optimal value for a given function, either a maximum or a minimum. It belongs to the heuristic category of optimization methods, concretely to swarm intelligence (SI). The used algorithm has a significant cost to compute, which is representative of the complex AI algorithms used in RTS video game research to define complex NPC behavior.

Jaya works as an iterative method. The objective function $f(x)$ has a vector x of n parameters which have to be tuned to minimize or maximize the function. In each iteration (k), a total of p candidate solutions, known as population, is generated. Every solution is a combination of x. Therefore, the whole population can be considered as a matrix of dimension (p, n). The best candidate of the iteration is the x candidate solution that obtains the best value of $f(x)$ (i.e., $f(x)_{best}$) out of all the population, while the worst candidate solution obtains the worst value of $f(x)$ (i.e., $f(x)_{worst}$) out of all the population. If $X_{j,k,i}$ is the value of the j^{th} variable for the k^{th} candidate during the i^{th} iteration, then this value is modified by means of the following equation:

$$X'_{j,k,i} = X_{j,k,i} + r_{1,j,i}(X_{j,best,i} - |X_{j,k,i}|) - r_{2,j,i}(X_{j,worst,i} - |X_{j,k,i}|) \quad (1)$$

where $X_{j,best,i}$ is the value of the j variable for the best candidate, and $X_{j,worst,i}$ is the value of the j variable for the worst candidate. In Eq. 1, $X'_{j,k,i}$ is the updated value of $X_{j,k,i}$, and $r_{1,j,i}(x_{j,best,i}-|X_{j,k,i}|)$ designates the tendency of the algorithm to move closer to the best solution, whereas the term $-r_{2,j,i}(X_{j,worst,i}-|X_{j,k,i}|)$ designates the tendency of the algorithm to avoid the worst solution. The new candidate $(X'_{j,k,i})$ is accepted only if it gives a better function evaluation. All the values accepted at the end of each iteration are maintained, so these values become the input for the next iteration. The algorithm is expected to move towards a better solution each iteration. Lastly, it is common to execute multiple runs of the algorithm with different random initial values.

The Jaya algorithm is well fit for a GPU implementation, as it has inherent parallel features that can be exploited. The GPU implementation of the algorithm has three levels of optimizations [82]:

- Independent execution level: Different independent executions of the algorithm are run concurrently.
- Candidate solution level: Each candidate solution is calculated concurrently.
- Objective function level: Some level of concurrency is used to calculate the value of the objective function. This level is only necessary if the objective function has a significant cost. Increased synchronization costs may hide any benefits from this level of optimization.

The AI algorithm in the game calculates the position of each soldier, trying to form two differentiated battalions. The positions are simplified to two of the three dimensions, because all NPCs are situated at the same height on a plane. Therefore, x represents the positions (in the horizontal and vertical axis) of the soldiers. The objective of the algorithm is to form two groups of NPCs, with as little distance between them as possible. In order to do so, the objective function $(f(x))$ calculates the squares for the two groups that form with the minimum and maximum value in both axis. Then it calculates the intersection area of the two group squares. The function returns the area of the intersection square.

In this work, a parallel version of Jaya has been implemented using the OpenCL library. It uses optimizations at the independent execution and candidate solution levels. Each candidate solution $(X_{k,i})$ is updated in a different thread. Then, a synchronization and a reduction operation is performed to find the best and worst solution of the iteration. Different runs of the algorithm with different initial random values are also executed at the same time in different work-groups. Greater parallelism could be achieved by updating each $X_{j,k,i}$ variable in a separate thread. This would involve optimizing at the objective function level. However, because the objective function is relatively simple, such optimization is not necessary. A limitation of this implementation is that the population number cannot be higher than the number of threads of a work group on the device the algorithm is running. This is due to the reduction operation, where memory is shared between the threads. Local memory is only accessible between threads of the same work group. In the systems the experiments were run, the maximum population possible was 256. The limit has not been a problem, as a population higher than 32 does not make the algorithm converge faster, nor does it obtain substantially worse solutions [83].

E. Thread management

The game loop has been multithreaded so different systems can run in parallel. Figure 5 is the diagram of the game loop. It uses two threads. The first one runs the render system, which has the task of executing the GPU render primitives to produce the image from the game data. The second thread is responsible for running the game logic systems that update the state of the game objects. The Input, AI, Collision, and Physics systems are run sequentially as there are data dependencies between them. Running them in parallel would not make a significant impact in performance for the chosen experiments, as besides the system in which the test focuses (AI or Collision), the rest carry no significant amount of computation. The reason behind this game loop design is to allow the game engine to use the GPUs and the CPU simultaneously. The dGPU is used to render the scene, while, simultaneously, the operations of the rest of the systems can be placed on the CPU, dGPU or iGPU. A synchronization step is put at the end of the loop to synchronize the threads before the next frame.

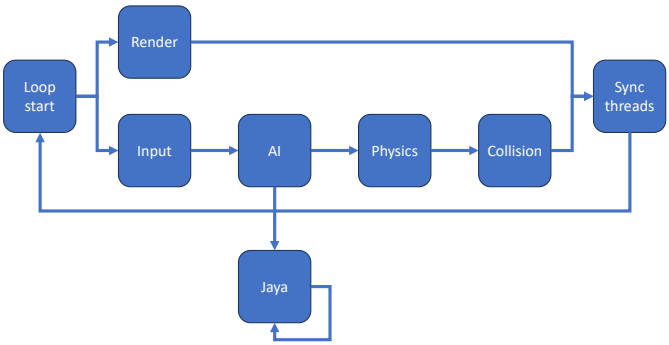


Fig. 5: Diagram of the multithreaded game loop

For the experiments, two types of tasks are modelled. The collision detection is a synchronous task that runs within the game loop. On the other hand, the Jaya AI algorithm runs asynchronously to the game loop, where the AI system checks and sets its state every loop iteration.

F. Frame time

The chosen game loop design defines how the different systems affect the final frame time. A frame is generated for every iteration of the game loop. Therefore, the frame time (T_f) behavior can be predicted using the subsequent formula:

$$T_f = max(T_r, T_s)$$
$$T_s = T_i + T_a + T_p + T_c \quad (2)$$

Where T_r is the time it takes to compute the render system, and T_s is the added time of all the game logic systems: Input (T_i), AI (T_a), Physics (T_p) and Collision (T_c).

The formula shows how the two tasks used in the experiments affect the frame time. The collision detection is a synchronous task within the game loop. It has a limited execution time before it starts reducing the performance of the system. That time is dependent on the computing time of the Render, Input, AI and Physics systems, and can be expressed as:

$$T_c \leq T_r - T_i - T_a - T_p \tag{3}$$

A collision detection computing time that exceeds that condition will make collisions the critical task of the game loop and will have a negative impact on the performance.

The Jaya AI task is asynchronous to the game loop, so its execution time can span several frames without affecting performance.

G. memory management

Memory has to be copied back and forth between main memory and GPU memory. However, this movement is specific to the render and GPU algorithms of the game.

The render needs to copy the texture and vertex data of the soldier 3D model, and the lightning information of the environment. To simulate a high rendering load, the soldier model is rendered the specified amount of times. If this were to be done using one drawing call for each drawn soldier, the shader and GPU memory configuration would cause a performance bottleneck in the system.

To address this issue, the instancing technique is used. It uses a single API call to draw multiple objects of the same model with different positions [84]. Instancing allows reducing the number of communications between the CPU and the GPU that happen for each drawing call. A consequence is that GPU resources have better utilization. Multiple drawing calls with low computing load are substituted with a single call with significant work. The outcome is more efficient rendering with a more stable behavior of the system due to the reduced CPU-GPU communication.

The collision detection algorithm has to copy to the GPU memory the physics components, which contain position, speed and rotation information, and the collision components, which contain bounding box information of each soldier. To retrieve the results, the physics components have to be copied back to main memory. This operation is performed once every frame, as the collision algorithm is a synchronous task within the game loop.

The Jaya AI algorithm variables are initialized randomly inside the GPU, so only a memory allocation operation is performed to store the variables for all the population. Once the algorithm finishes, Only the best individual solution from all runs is copied to main memory to get the results. The copy has a size equal to the number of variables. The frequency of these memory movements does not have to happen every game loop iteration and depends on the time it takes for the algorithm to compute, and be restarted.

The memory movement in and out of the GPU can have a performance impact in the game. The impact is different for dGPUs and iGPUs. The iGPU is connected to the main memory (RAM) via the PCI Express socket. The speed of a memory transfer on the PCI is slower than CPU main memory access, so it acts as a bottleneck. On the other hand, the iGPU is connected to the CPU ring, and has direct access to the main memory. It is able to do "zero" copy buffers, that is, directly access data allocated in main memory. This means, that GPU memory copies are slower on dGPUs than iGPUs [85]. On the other hand, iGPU memory access can be negatively impacted by high CPU memory usage and vice versa.

V. RESULTS

The first thing that is going to be discussed, is how a computing kernel interferes with the graphics computations, and how the use of the iGPU improves performance. To delve into low level knowledge on how the dGPU is behaving in each scenario, in the first experiment, Nvidia Nsight Graphics is used to trace the load on the dGPU when rendering a frame of the game. Figure 6 shows the Nsight traces for both a single GPU (dGPU) architecture, and the hybrid (dGPU + iGPU) architecture. The game is using an OpenCL kernel to compute the AI Jaya algorithm.

The first thing that the trace shows, is that the GPU executes different tasks sequentially, even though they might be submitted at the same time. If looked closely, in the GPU Context timeline, the OpenGL Context (in green) gets interrupted to perform other tasks, even though the "glDraw" calls have not finished yet. The gray GPU context is the OpenCL computing context, and the red one is other contexts outside the application domain. This phenomenon can also be checked using the SM Occupancy timeline. The Streaming Multiprocessor (SM) is a hardware resource of the Nvidia GPUs that is capable of executing multiple threads, in hardware groups called warps. The gray color represents the accumulative percentage of unused warps in active SMs. The blue color is the percentage of active vertex/tessellation/geometry warps. The green color is for the pixel warps. Lastly, the yellow color represents the compute warps. Looking at that timeline, it can be seen that the graphic computation (vertex/tessellation/geometry and pixel warps) gets interrupted to introduce compute oriented work (compute warps). The GPU scheduler assigns a time window a process to be executed, before giving way to another process. The programmer has no control over it. Therefore, having multiple processes executing at the same time on the GPU is a source of instability and unpredictability. The scheduler may take different scheduling decisions on different devices and environments [86].

However, computation is not the only area where the drawing calls are affected. Swapping the GPU context to perform other tasks introduces memory overhead. Different applications need their own memory, that needs to be swapped in and out of the memory hierarchy. The L2 read hit rates are impacted by changing the contexts. Also, the spikes in PCIe Bandwidth are higher, which represent the amount of memory that is transferred in and out the GPU. All that makes the frame time higher in the single GPU architecture.

The next step is to test the performance obtained with the prototype game using the hybrid GPU architecture. In the

This article has been accepted for publication in IEEE Transactions on Games. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TG.2025.3533949

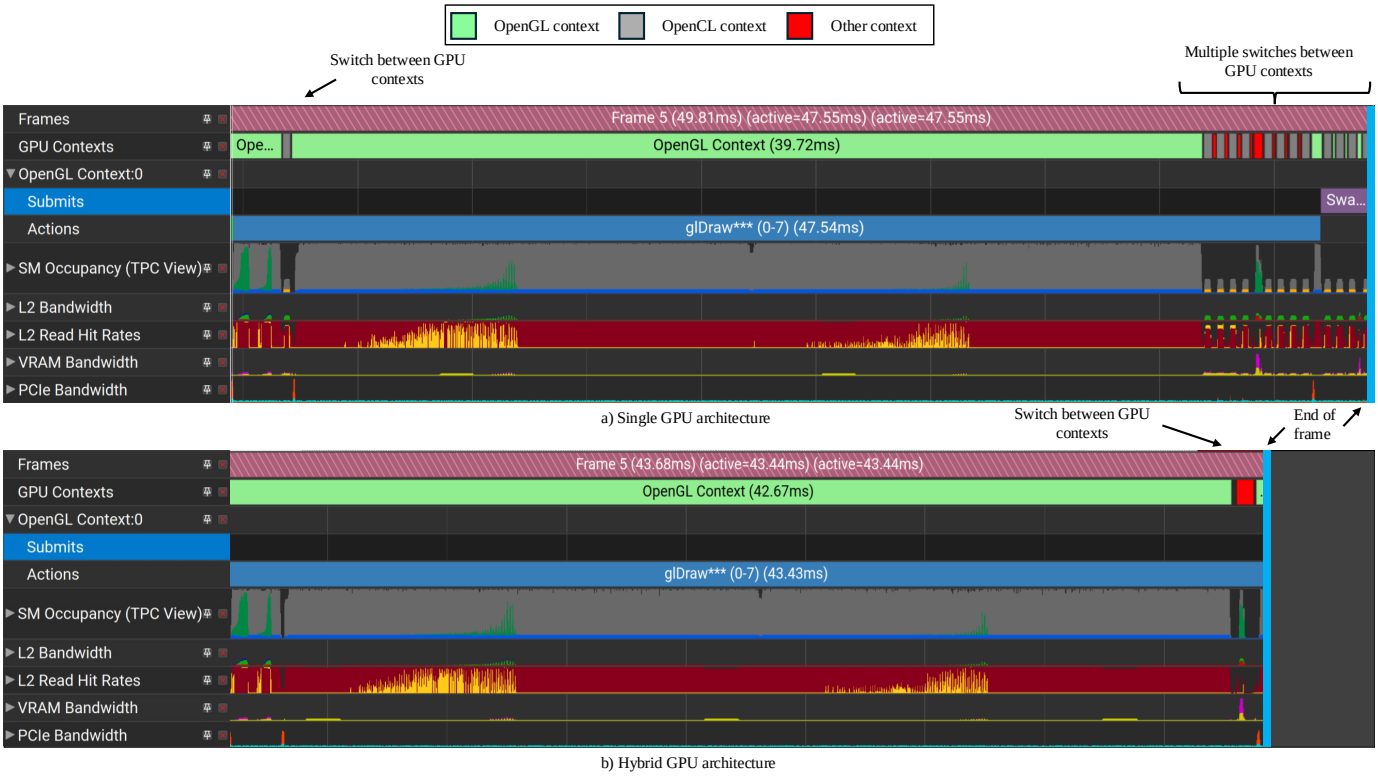


Fig. 6: Nvidia Nsight trace of the dGPU during a single frame. Left legend: timeline aspects; top legend: GPU context colors.

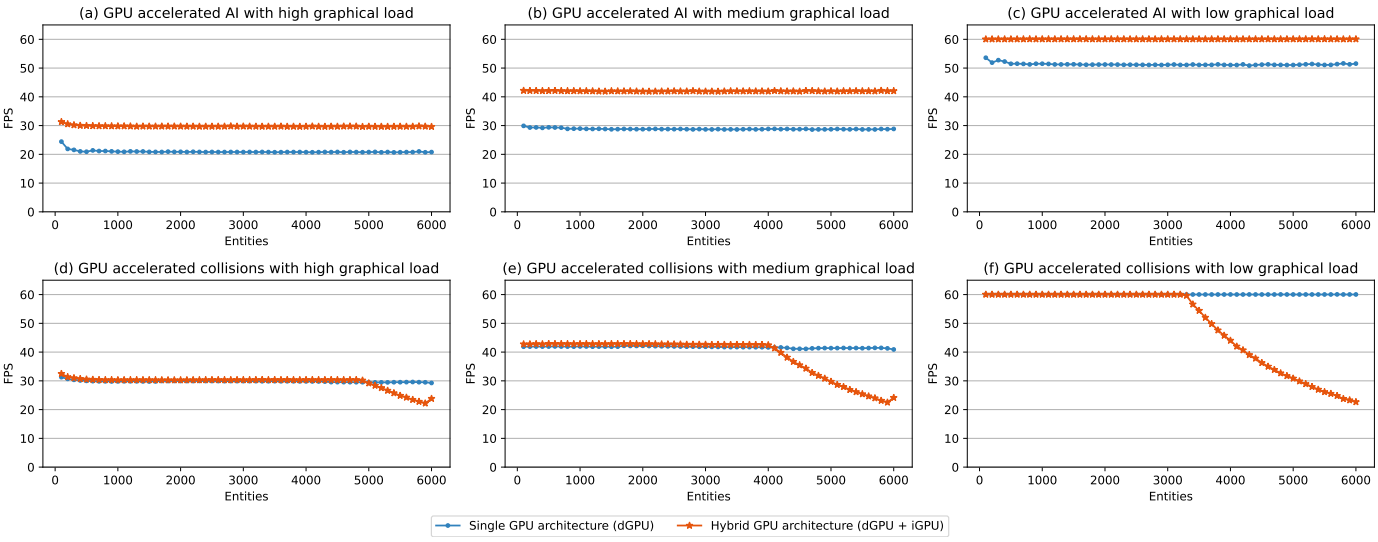


Fig. 7: Performance (FPS) comparison of the engine on a high-end system

following experiments, the AI and Collision systems each perform GPU computations, but not simultaneously. Game performance is compared for each system individually, with their GPU computations tested on both the dGPU and iGPU. The experiments are run in different systems, representing a high-end PC, and a low-end PC. A high-end system represents a computer with high-quality, top-of-the-line commodity hardware components. A low-end system represents a computer with a lower budged that uses a lower quality, older commodity hardware. The exact hardware components of each system are defined at the beginning of each experiment.

For the measurements, the average frame time is used to compute the average Frames per Seconds (FPS) ratio. A total of, at least, 500 frames are averaged, and the first 100 frames are discarded due to initial system instability. In the tests, three main scenarios are shown based on the amount of rendering load. The load is tuned to the desired amount where the prototype runs close to the target frame rate, and maintained through the experiment. High is when the game is running at 60 FPS, medium scenario runs at 45 FPS, and low runs at 30 FPS. In all tests, a screen resolution of 1920×1080 has been used. The Jaya AI algorithm has the fixed parameters of 100

iterations, 15 runs, and a population of 256. These parameters were chosen to be compatible with both test systems. The number of variables varies with the number of entities of the test. We provide the code used to perform the experiments in a public github repository[1].

A. High-end PC experiments

The first system where the experiments were conducted represents a high-end PC. Figure 7 shows the results in FPS. It has an Intel i7-9700K CPU with Intel UHD 630 iGPU, 32 GB of RAM DDR4, Nvidia TITAN RTX dGPU with 24 GB of VRAM, and runs 64-bit Windows 10 OS.

In this system, the benefits of using the iGPU are shown in plots 7.a, 7.b, and 7.c, when the engine uses GPU accelerated AI. The GPU AI computations interfere with the graphic computations when run on the dGPU, worsening the performance. Performance is better when using the iGPU, as both the AI and render computations can run independently and do not interfere with each other.

However, for the collision computations, shown in plots 7.d, 7.e, and 7.f, the speedup obtained is much smaller, and a sudden drop in performance is observed after a certain point. In this system, the difference in computing power between the Intel UHD 630 and the Nvidia TITAN RTX is so significant that for the same amount of objects with collision detection, the iGPU takes much longer than the dGPU to complete the task.

The sudden drop in performance when using the iGPU to compute the collisions happens where collision detection exceeds the critical point shown in Formula 3. At that point, the collision computations become the critical task of the game loop that determines the frame time. As the number of entities keep increasing, collision performance worsens on the hybrid GPU architecture at a faster rate than the graphics computations do in the single GPU architecture.

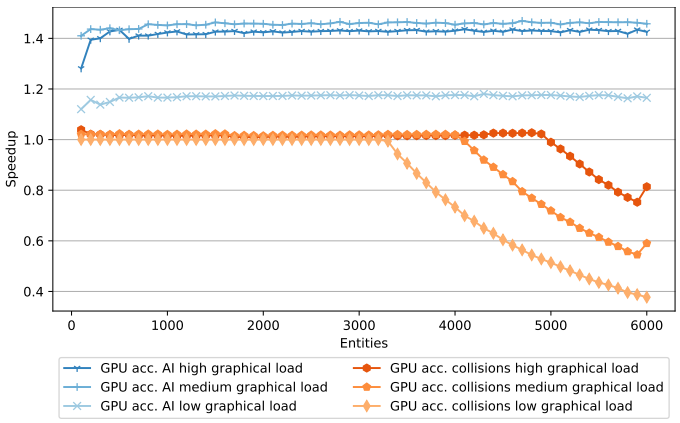


Fig. 8: Performance (Speedup) comparison of the engine on a high-end system

Figure 8 shows the speedups obtained with the hybrid architecture on the high-end PC. The gain in performance is more clearly seen. The FPS goes up to a 46.9% when using the hybrid architecture with the AI algorithm. On the other hand, the performance improves a maximum of 3.8% in the collision detection experiment.

Figure 9 illustrates the execution times of the Jaya AI algorithm on both single and hybrid GPU architectures. The results indicate that, despite the significant difference in computing power between the Intel UHD 630 and the Nvidia TITAN RTX, the penalty of executing the rendering load on the same device makes the execution times worse in the dGPU of the single GPU architecture than the iGPU of the hybrid architecture.

B. Low-end PC experiments

The second test system represents a modest setup. The results, in FPS, can be seen in Figure 10. This system has an Intel i5-3570K CPU with Intel HD 4000 iGPU, 16 GB of RAM DDR3, Nvidia GTX 960 dGPU with 2 GB of VRAM, and runs 64-bit Windows 10 OS.

In this setup, the benefits of using the iGPU are shown in every test, even with the collision detection system. Performance is improved by using the proposed technique in this system. The higher the GPU load, the better the results obtained.

It is worth noting that when the Jaya workload increases, when incrementing the number of entities, the GPU scheduler seems to first prioritize the graphics computations. Several frames occur with a low rendering time, followed by a slow frame, when the scheduler prioritizes the Jaya computations. The higher the number of entities, the higher the number of "fast" frames, followed by a proportionally slower frame. This behaviour leads to the apparent increase in performance, for the single GPU architecture with medium graphical load (Figure 10.b), when increasing the number of entities. The number of normal frames compensates for the slower frame. When averaging the FPS measure, the average is higher. This behaviour is not present in the high-end system, where the penalty of the computing load is spread evenly across all frames.

In the low-end system, the collisions experiments show better performance for the hybrid architecture until the collision algorithm becomes the critical task of the game loop.

The speedups in Figure 11 show the techniques used obtain a peak performance increase of 4572% in the modest system, for AI with high graphical load. The maximum performance improvement in the collisions experiment is of 16.4%, also with high graphical load.

Performance is improved when a significant computing load is present. The experiments show that the proposed hybrid architecture benefits will be more noticeable in more modest systems. The reason is that the difference in computing power between the dGPU and the iGPU is lower than in high-end systems. In low-end environments, the improvements made with the hybrid architecture can be seen in a wider range of circumstances. The experiments have also shown that the technique has to be fine-tuned for the specific algorithm and the set of devices where it is expected to run in order to get the best performance out of it.

[1]https://github.com/aml125/HybridArchitecture

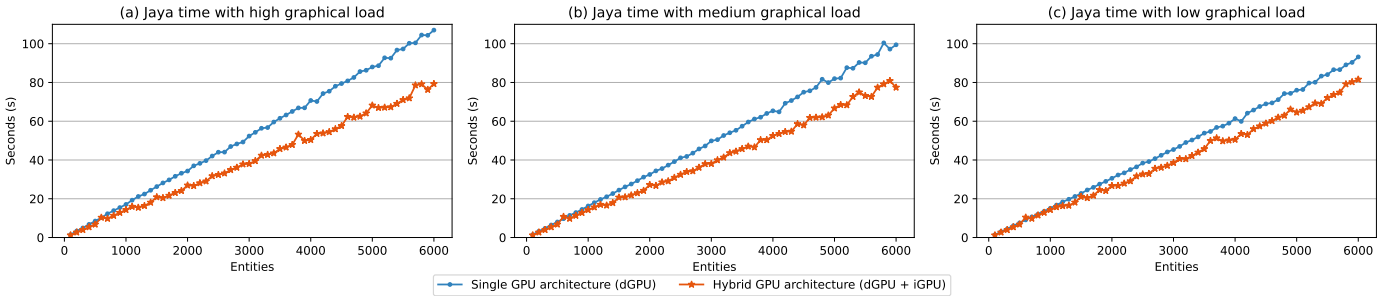


Fig. 9: Execution times of the Jaya AI algorithm in the high-end system

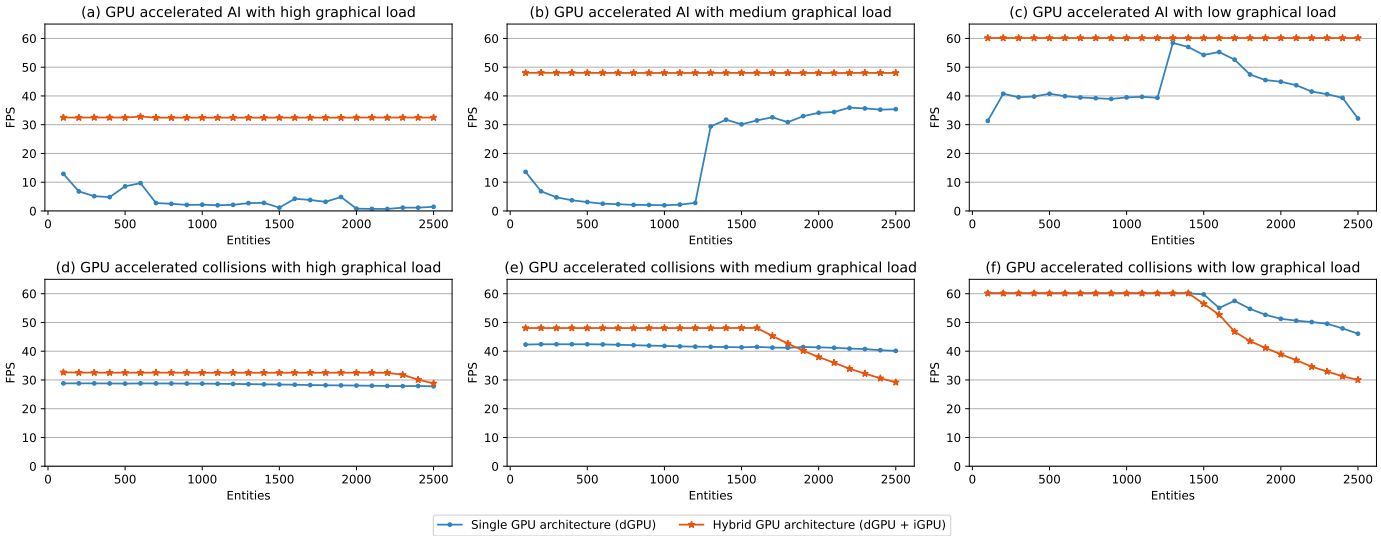


Fig. 10: Performance (FPS) comparison of the engine on a low-end system

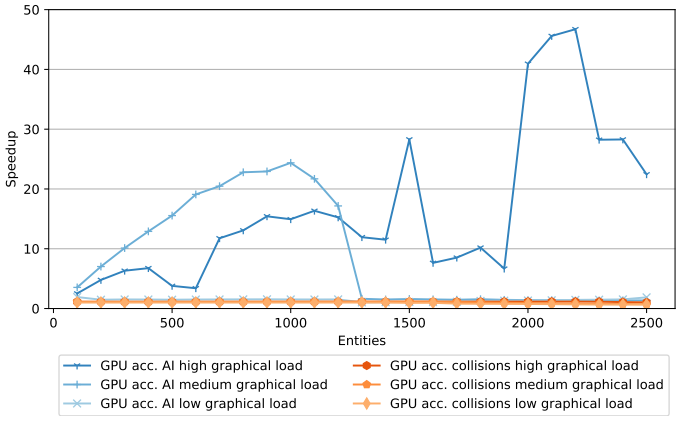


Fig. 11: Performance (Speedup) comparison of the engine on a low-end system

VI. Conclusions

There is the need in video games to obtain the best performance possible. In the field of RTS games, the simultaneous presence of big amounts of units can cause performance issues. Some of the tasks that have to be done to every entity, such as AI and collision detection tasks, are highly parallel in nature, and therefore subject to be accelerated via GPGPU computations. However, the performance gains are limited as the main GPU is already used to render the game. In response, a hybrid architecture where the game engine makes use of the dGPU and the iGPU at the same time has been tested.

Based on the hybrid architecture model, a custom game engine has been built. It is able to use the CPU, dGPU and iGPU at the same time to improve resource usage and performance. On top of that, a prototype game has been developed. It has thousands of NPC units with expensive AI and Physics computations, aimed to model the characteristics of RTS games. The collision detection and AI tasks are accelerated using GPGPU computations, and can be moved freely to be executed in all available computing devices.

The experiments show that the use of the hybrid architecture improves the performance in games with large amounts of entities. Performance can be improved when the amount of work the GPU has to do is significant. Unloading the dGPU of the GPGPU computations decreases the frame rendering time, while at the same time increases the performance of the rest of the systems by enabling them to use a GPU device. However, the difference in capabilities between the dGPU and the iGPU has to be taken into account, specially in time constrained tasks within the main game loop. The architecture needs to be tuned considering the tasks that are involved, the algorithms used, and the range of devices it is expected to run.

This study has several limitations that need to be discussed. First, the use of the proposed hybrid architecture

is only feasible in computers where both an iGPU and a dGPU are present. Furthermore, the study has been made using OpenGL for rendering and OpenCL for the GPGPU computations. There are other technologies that can be used for those tasks, such as Vulkan and Direct3D for rendering, and computing shaders for GPGPU. Also, the experiments are made on a custom game engine and test game, and the experiments are performed on a specific case where three frame rate scenarios and only one screen resolution are studied. Furthermore, the collision detection is not using the best optimization possible. In production systems, collisions may be further optimized using axis aligned collisions or quadtrees. Other potential rendering and programming optimizations were not explored in this study. The main focus is to test the hybrid architecture and analyze its characteristics and performance. Additional optimizations could further enhance performance, provided the conditions for performance gains are maintained. Finally, this study has only examined a hybrid architecture where computations are allocated exclusively to either the dGPU or the iGPU.

Future work would be beneficial to broaden the scope of this study. There are technologies not used in this work, such as compute shaders, that could be implemented to compare the performance against the hybrid architecture with OpenGL and OpenCL presented in this work. Moreover, tests of the hybrid architecture could be made in real world games. The hybrid architecture could be improved to leverage work between the iGPU and the dGPU in real-time, instead of using only one of the devices for GPGPU computations. Lastly, the difference in capabilities between the dGPU and the iGPU should be quantified to simplify implementations of the hybrid architecture in real applications. For that, strategies such as measuring the difference in overall thread parallelism, and comparing memory copy times to GPU memory could be employed.

Overall, the presented architecture brings the opportunity to improve the performance of games with large numbers of entities, such as RTS games, by using all the resources available in the system. It improves the gameplay experience of players as performance is improved without upgrading the hardware the game runs on. It allows the use of more complex and resource intensive AI and physics algorithms. It can have a positive impact in the industry by allowing the use in video games of bigger numbers of NPCs with more complex behaviors.

ACKNOWLEDGMENT

This work was supported by the Spanish Research Agency (AEI) (DOI: 10.13039/501100011033) under project Serverless4HPC PID2023-152804OB-I00.

REFERENCES

[1] Statista, "Video Games - Worldwide | Statista Market Forecast," 2024. [Online]. Available: https://www.statista.com/outlook/dmo/digital-media/video-games/worldwide

[2] D. A. Chugh, D. C. Jain, and D. A. Kumar, "Design of Artificial Intelligence Enabled Game Engine," in *Proceedings of the 4th International Conference on Information Management & Machine Intelligence*, ser. ICIMMI '22. New York, NY, USA: Association for Computing Machinery, May 2023, pp. 1–5.

[3] G. Koulaxidis and S. Xinogalos, "Improving Mobile Game Performance with Basic Optimization Techniques in Unity," *Modelling*, vol. 3, no. 2, pp. 201–223, Jun. 2022, number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[4] S. Cass, "Mind games [computer game AI]," *IEEE Spectrum*, vol. 39, no. 12, pp. 40–44, Dec. 2002, conference Name: IEEE Spectrum.

[5] G. Robertson and I. Watson, "A Review of Real-Time Strategy Game AI," *AI Magazine*, vol. 35, no. 4, pp. 75–104, Dec. 2014, number: 4.

[6] D. Long, B. Morkos, and S. Ferguson, "Toward Quantifiable Evidence of Excess' Value Using Personal Gaming Desktops," *Journal of Mechanical Design*, vol. 143, no. 031712, Jan. 2021.

[7] H. Chen, S. Gao, H.-C. Yeh, and X. Sun, "Evolution of the pc industry and role of data analytics," in *2022 4th International Conference on Economic Management and Cultural Industry (ICEMCI 2022)*. Atlantis Press, 2022, pp. 1979–1989.

[8] Y. Shan, Y. Yang, X. Qian, and Z. Yu, "Guser: A GPGPU Power Stressmark Generator," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Mar. 2024, pp. 1111–1124, iSSN: 2378-203X.

[9] C. Stéphane and J. Éric, "Binary GPU-Planning for Thousands of NPCs," in *2020 IEEE Conference on Games (CoG)*, Aug. 2020, pp. 678–681, iSSN: 2325-4289.

[10] K. Mahale, S. Kanaskar, P. Kapadnis, M. Desale, and S. M. Walunj, "Acceleration of game tree search using GPGPU," in *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, Oct. 2015, pp. 550–553.

[11] C. McMillan, E. Hart, and K. Chalmers, "Collaborative Diffusion on the GPU for Path-Finding in Games," in *Applications of Evolutionary Computation*, ser. Lecture Notes in Computer Science, A. M. Mora and G. Squillero, Eds. Cham: Springer International Publishing, 2015, pp. 418–429.

[12] Y. Zhou and J. Zeng, "Massively Parallel A* Search on a GPU," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, Feb. 2015, number: 1.

[13] 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 *ACM SIGGRAPH 2008 Games*, ser. SIGGRAPH '08. New York, NY, USA: Association for Computing Machinery, Aug. 2008, pp. 52–101.

[14] A. Greß, M. Guthe, and R. Klein, "GPU-based Collision Detection for Deformable Parameterized Surfaces," *Computer Graphics Forum*, vol. 25, no. 3, pp. 497–506, 2006.

[15] N. K. Govindaraju, M. C. Lin, and D. Manocha, "Fast and reliable collision detection using graphics processors," in *Proceedings of the twenty-first annual symposium on Computational geometry*, ser. SCG '05. New York, NY, USA: Association for Computing Machinery, Jun. 2005, pp. 384–385.

[16] C. Zeller, "Cloth simulation on the GPU," in *ACM SIGGRAPH 2005 Sketches on - SIGGRAPH '05*. Los Angeles, California: ACM Press, 2005, p. 39.

[17] A. S. Aquilio, J. C. Brooks, Y. Zhu, and G. S. Owen, "Real-Time GPU-Based Simulation of Dynamic Terrain," in *Advances in Visual Computing*, ser. Lecture Notes in Computer Science, G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, A. Nefian, G. Meenakshisundaram, V. Pascucci, J. Zara, J. Molineros, H. Theisel, and T. Malzbender, Eds. Berlin, Heidelberg: Springer, 2006, pp. 891–900.

[18] S. Rødal, G. Storli, and O. E. Gundersen, "Physically Based Simulation and Visualization of Fire in Real-Time using the GPU," *Theory and Practice of Computer Graphics*, 2006.

[19] J. R. Monfort and M. Grossman, "Scaling of 3D game engine workloads on modern multi-GPU systems," in *Proceedings of the Conference on High Performance Graphics 2009*, ser. HPG '09. New York, NY, USA: Association for Computing Machinery, Aug. 2009, pp. 37–46.

[20] X. Ren and M. Lis, "CHOPIN: Scalable Graphics Rendering in Multi-GPU Systems via Parallel Image Composition," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2021, pp. 709–722, iSSN: 2378-203X.

[21] D. Sekar and B. Chinnakrishnan, "Integration of Graphics Processing Cores with Microprocessors," in *Design of 3D Integrated Circuits and Systems*, D. C. Sekar and B. Chinnakrishnan, Eds. CRC Press, 2015, num Pages: 14.

[22] P. Gera, H. Kim, H. Kim, S. Hong, V. George, and C.-K. Luk, "Performance Characterisation and Simulation of Intel's Integrated GPU Architecture," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2018, pp. 139–148.

[23] T. Rath and N. Preethi, "Application of AI in Video Games to Improve Game Building," in *2021 10th IEEE International Conference on*

Communication Systems and Network Technologies (CSNT), Jun. 2021, pp. 821–824, iSSN: 2329-7182.

[24] V. Levytskyi, M. Tsiutsiura, A. Yerukaiev, N. Rusan, and T. Li, "The Working Principle of Artificial Intelligence in Video Games," in *2023 IEEE International Conference on Smart Information Systems and Technologies (SIST)*, May 2023, pp. 246–250.

[25] A. Hubble, J. Moorin, and A. S. Khuman, "Artificial Intelligence in FPS Games: NPC Difficulty Effects on Gameplay," in *Fuzzy Logic: Recent Applications and Developments*, J. Carter, F. Chiclana, A. S. Khuman, and T. Chen, Eds. Cham: Springer International Publishing, 2021, pp. 165–190.

[26] R. Ishii, S. Ito, R. Thawonmas, and T. Harada, "A Fighting Game AI Using Highlight Cues for Generation of Entertaining Gameplay," in *2019 IEEE Conference on Games (CoG)*, Aug. 2019, pp. 1–6, iSSN: 2325-4289.

[27] I. Millington, *Artificial intelligence for games*, ser. The Morgan Kaufmann series in interactive 3D technology. Amsterdam Heidelberg: Elsevier Morgan Kaufmann, 2006.

[28] C. Fairclough, M. Fagan, B. Mac Namee, and P. Cunningham, "Research directions for ai in computer games," *Trinity College Dublin, Department of Computer Science*, 2001.

[29] R. Lara-Cabrera, C. Cotta, and A. J. Fernández-Leiva, "A review of computational intelligence in RTS games," in *2013 IEEE Symposium on Foundations of Computational Intelligence (FOCI)*, Apr. 2013, pp. 114–121.

[30] M. Buro, "Real-time strategy games: A new ai research challenge," in *IJCAI*, 2003, pp. 1534–1535.

[31] T. Fujiki, K. Ikeda, and S. Viennot, "A platform for turn-based strategy games, with a comparison of monte-carlo algorithms," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2015, pp. 407–414.

[32] M. Buro and T. M. Furtak, "Rts games and real-time ai research," in *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, vol. 6370, 2004, pp. 1–8.

[33] G. Robertson, "Applying Learning by Observation and Case-Based Reasoning to Improve Commercial RTS Game AI," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 8, no. 6, pp. 31–33, 2012, number: 6.

[34] A. Khan, K. Yang, Y. Fu, F. Lou, W. Jifara, F. Jiang, and L. Shaohui, "A Competitive Combat Strategy and Tactics in RTS Games AI and StarCraft," in *Advances in Multimedia Information Processing – PCM 2017*, B. Zeng, Q. Huang, A. El Saddik, H. Li, S. Jiang, and X. Fan, Eds. Cham: Springer International Publishing, 2018, pp. 3–12.

[35] D. Churchill, A. Saffidine, and M. Buro, "Fast Heuristic Search for RTS Game Combat Scenarios," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 8, no. 1, pp. 112–117, 2012, number: 1.

[36] P.-A. Andersen, M. Goodwin, and O.-C. Granmo, "Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug. 2018, pp. 1–8, iSSN: 2325-4289.

[37] S. Wender and I. Watson, "Combining Case-Based Reasoning and Reinforcement Learning for Unit Navigation in Real-Time Strategy Game AI," in *Case-Based Reasoning Research and Development*, ser. Lecture Notes in Computer Science, L. Lamontagne and E. Plaza, Eds. Cham: Springer International Publishing, 2014, pp. 511–525.

[38] C. Lucero, C. Izumigawa, K. Frederiksen, L. Nans, R. Iden, and D. S. Lange, "Human-Autonomy Teaming and Explainable AI Capabilities in RTS Games," in *Engineering Psychology and Cognitive Ergonomics. Cognition and Design*, ser. Lecture Notes in Computer Science, D. Harris and W.-C. Li, Eds. Cham: Springer International Publishing, 2020, pp. 161–171.

[39] T. Machalewski, M. Marek, and A. Ochmann, "Optimization of Parameterized Behavior Trees in RTS Games," in *Artificial Intelligence and Soft Computing*, ser. Lecture Notes in Computer Science, L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz, and J. M. Zurada, Eds. Cham: Springer International Publishing, 2023, pp. 387–398.

[40] G. Lorthioir and K. Inoue, "Design Adaptive AI for RTS Game by Learning Player's Build Order," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. Yokohama, Japan: International Joint Conferences on Artificial Intelligence Organization, Jul. 2020, pp. 5194–5195.

[41] T. Liu, Z. Zheng, H. Li, K. Bian, and L. Song, "Playing Card-Based RTS Games with Deep Reinforcement Learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. Macao, China: International Joint Conferences on Artificial Intelligence Organization, Aug. 2019, pp. 4540–4546.

[42] S. Risi and M. Preuss, "From Chess and Atari to StarCraft and Beyond: How Game AI is Driving the World of AI," *KI - Künstliche Intelligenz*, vol. 34, no. 1, pp. 7–17, Mar. 2020.

[43] R.-Z. Liu, "Rethinking of AlphaStar," Sep. 2021, arXiv:2108.03452 [cs].

[44] K. Arulkumaran, A. Cully, and J. Togelius, "AlphaStar: an evolutionary computation perspective," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. Prague Czech Republic: ACM, Jul. 2019, pp. 314–315.

[45] C. Berner, G. Brockman, B. Chan, V. Cheung, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with Large Scale Deep Reinforcement Learning," *OpenAI*, 2019.

[46] J. P. Sousa, R. Tavares, J. P. Gomes, and V. Mendonça, "Review and analysis of research on video games and artificial intelligence: a look back and a step forward," *Procedia Computer Science*, vol. 204, pp. 315–323, 2022.

[47] D. B. Kirk and W. H. Wen-Mei, *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann, 2016.

[48] K. Shafie Khorassani, J. Hashmi, C.-H. Chu, C.-C. Chen, H. Subramoni, and D. K. Panda, "Designing a ROCm-Aware MPI Library for AMD GPUs: Early Experiences," in *High Performance Computing*, B. L. Chamberlain, A.-L. Varbanescu, H. Ltaief, and P. Luszczek, Eds. Cham: Springer International Publishing, 2021, pp. 118–136.

[49] A. Munshi, B. Gaster, T. G. Mattson, and D. Ginsburg, *OpenCL programming guide*. Pearson Education, 2011.

[50] S. I. Gunadi and P. Yugopuspito, "Real-Time GPU-based SPH Fluid Simulation Using Vulkan and OpenGL Compute Shaders," in *2018 4th International Conference on Science and Technology (ICST)*, Aug. 2018, pp. 1–6.

[51] D. Geer, "Vendors Upgrade Their Physics Processing to Improve Gaming," *Computer*, vol. 39, no. 8, pp. 22–24, Aug. 2006, conference Name: Computer.

[52] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, May 2008, conference Name: Proceedings of the IEEE.

[53] M. Joselli, C. N. Vasconcelos, and E. Clua, "A New Architecture for Games and Simulations Using GPUs," *INFORMATION TECHNOLOGY IN INDUSTRY*, vol. 2, no. 1, Mar. 2014, number: 1.

[54] J.-H. Nah, J.-W. Kim, J. Park, W.-J. Lee, J.-S. Park, S.-Y. Jung, W.-C. Park, D. Manocha, and T.-D. Han, "HART: A Hybrid Architecture for Ray Tracing Animated Scenes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 21, no. 3, pp. 389–401, Mar. 2015, conference Name: IEEE Transactions on Visualization and Computer Graphics.

[55] M. Tayyub and G. N. Khan, "Heterogeneous design and efficient cpu-gpu implementation of collision detection," *IADIS International Journal on Computer Science and Information Systems*, vol. 14, no. 2, pp. 25–40, 2019.

[56] M. Joselli, M. Zamith, E. Clua, A. Montenegro, R. Leal-Toledo, A. Conci, P. Pagliosa, L. Valente, and B. Feijó, "An adaptative game loop architecture with automatic distribution of tasks between CPU and GPU," *Computers in Entertainment*, vol. 7, no. 4, pp. 50:1–50:15, Jan. 2010.

[57] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *IEEE Network*, vol. 27, no. 4, pp. 16–21, Jul. 2013, conference Name: IEEE Network.

[58] V. Sánchez Ribes, H. Mora, A. Sobecki, and F. J. Mora Gimeno, "Mobile Cloud computing architecture for massively parallelizable geometric computation," *Computers in Industry*, vol. 123, p. 103336, Dec. 2020.

[59] D. Sharma, A. Badal, and A. Badano, "hybrid: a CPU–GPU Monte Carlo method for modeling indirect x-ray detectors with columnar scintillators," *Physics in Medicine & Biology*, vol. 57, no. 8, p. 2357, Apr. 2012, publisher: IOP Publishing.

[60] M. A. Zidan, T. Bonny, and K. N. Salama, "High performance technique for database applications using a hybrid GPU/CPU platform," in *Proceedings of the 21st edition of the great lakes symposium on Great lakes symposium on VLSI*, ser. GLSVLSI '11. New York, NY, USA: Association for Computing Machinery, May 2011, pp. 85–90.

[61] M. Papadrakakis, G. Stavroulakis, and A. Karatarakis, "A new era in scientific computing: Domain decomposition methods in hybrid CPU–GPU architectures," *Computer Methods in Applied Mechanics and Engineering*, vol. 200, no. 13, pp. 1490–1508, Mar. 2011.

[62] Z. Zhu, S. Xu, J. Tang, and M. Qu, "GraphVite: A High-Performance CPU-GPU Hybrid System for Node Embedding," in *The World Wide*

Web Conference, ser. WWW '19. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 2494–2504.

[63] S. Chen and X. Li, "A hybrid GPU/CPU FFT library for large FFT problems," in *2013 IEEE 32nd International Performance Computing and Communications Conference (IPCCC)*, Dec. 2013, pp. 1–10, iSSN: 2374-9628.

[64] S. Tomov, J. Dongarra, and M. Baboulin, "Towards dense linear algebra for hybrid GPU accelerated manycore systems," *Parallel Computing*, vol. 36, no. 5, pp. 232–240, Jun. 2010.

[65] G. Lupescu, E.-I. Sluşanschi, and N. Tăpuş, "Using the Integrated GPU to Improve CPU Sort Performance," in *2017 46th International Conference on Parallel Processing Workshops (ICPPW)*, Aug. 2017, pp. 39–44, iSSN: 1530-2016.

[66] M. Kuhrt, M. Körber, and B. Seeger, "iGPU-Accelerated Pattern Matching on Event Streams," in *Data Management on New Hardware*, ser. DaMoN'22. New York, NY, USA: Association for Computing Machinery, Jun. 2022, pp. 1–7.

[67] R. Campos, D. Marques, S. Santander-Jiménez, L. Sousa, and A. Ilic, "Heterogeneous CPU+iGPU Processing for Efficient Epistasis Detection," in *Euro-Par 2020: Parallel Processing*, ser. Lecture Notes in Computer Science, M. Malawski and K. Rzadca, Eds. Springer International Publishing, 2020, pp. 613–628.

[68] G. Lupescu and N. Ţăpuş, "Design of hashtable for heterogeneous architectures," in *2021 23rd International Conference on Control Systems and Computer Science (CSCS)*, May 2021, pp. 172–177, iSSN: 2379-0482.

[69] J. Tseng, R. Wang, J. Tsai, Y. Wang, and T.-Y. C. Tai, "Accelerating Open vSwitch with Integrated GPU," in *Proceedings of the Workshop on Kernel-Bypass Networks*, ser. KBNets '17. New York, NY, USA: Association for Computing Machinery, Aug. 2017, pp. 7–12.

[70] T. Karnagel, M. Hille, M. Ludwig, D. Habich, W. Lehner, M. Heimel, and V. Markl, "Demonstrating efficient query processing in heterogeneous environments," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: Association for Computing Machinery, Jun. 2014, pp. 693–696.

[71] G. Lupescu, L. Gheorghe, and N. Tapus, "Commodity hardware performance in AES processing," in *2014 IEEE 13th International Symposium on Parallel and Distributed Computing*, Jun. 2014, pp. 82–86, iSSN: 2379-5352.

[72] E. Peek, B. Wünsche, and C. Lutteroth, "Using Integrated GPUs to Perform Image Warping for HMDs," in *Proceedings of the 29th International Conference on Image and Vision Computing New Zealand*, ser. IVCNZ '14. New York, NY, USA: Association for Computing Machinery, Nov. 2014, pp. 172–177.

[73] M. Muratet and D. Garbarini, "Accessibility and Serious Games: What About Entity-Component-System Software Architecture?" in *Games and Learning Alliance*, ser. Lecture Notes in Computer Science, I. Marfisi-Schottman, F. Bellotti, L. Hamon, and R. Klemke, Eds. Cham: Springer International Publishing, 2020, pp. 3–12.

[74] S. Bilas, "A data-driven game object system," in *Game Developers Conference Proceedings*, 2002.

[75] P. Lange, R. Weller, and G. Zachmann, "Wait-free hash maps in the entity-component-system pattern for realtime interactive systems," in *2016 IEEE 9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, Mar. 2016, pp. 1–8, iSSN: 2328-7829.

[76] K. Maystrenko, AnthonyMyers, and Danimal, "Animated defender aka brigand," Dec 2014. [Online]. Available: https://opengameart.org/content/animated-defender-aka-brigand

[77] T. Koziara and N. Bicanic, "Bounding box collision detection," in *13th acme conference: university of sheffield*, 2005.

[78] K. A. Rani, *Learning Unity Physics*. Packt Publishing, 2014.

[79] P. Cai, C. Indhumathi, Y. Cai, J. Zheng, Y. Gong, T. S. Lim, and P. Wong, "Collision Detection Using Axis Aligned Bounding Boxes," in *Simulations, Serious Games and Their Applications*, ser. Gaming Media and Social Effects, Y. Cai and S. L. Goei, Eds. Singapore: Springer, 2014, pp. 1–14.

[80] K. Hegeman, N. A. Carr, and G. S. P. Miller, "Particle-Based Fluid Simulation on the GPU," in *Computational Science – ICCS 2006*, ser. Lecture Notes in Computer Science, V. N. Alexandrov, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, Eds. Berlin, Heidelberg: Springer, 2006, pp. 228–235.

[81] R. Rao, "Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems," *International Journal of Industrial Engineering Computations*, vol. 7, no. 1, pp. 19–34, 2016.

[82] A. Jimeno-Morenilla, J.-L. Sánchez-Romero, H. Migallón, and H. Mora-Mora, "Jaya optimization algorithm with GPU acceleration," *The Journal of Supercomputing*, vol. 75, no. 3, pp. 1094–1106, 2019, publisher: Springer US.

[83] H. Rico-Garcia, J.-L. Sanchez-Romero, A. Jimeno-Morenilla, H. Migallon-Gomis, H. Mora-Mora, and R. V. Rao, "Comparison of High Performance Parallel Implementations of TLBO and Jaya Optimization Methods on Manycore GPU," *IEEE Access*, vol. 7, pp. 133 822–133 831, 2019, conference Name: IEEE Access.

[84] M. Lujan, M. McCrary, B. W. Ford, and Z. Zong, "Vulkan vs OpenGL ES: Performance and Energy Efficiency Comparison on the big.LITTLE Architecture," in *2021 IEEE International Conference on Networking, Architecture and Storage (NAS)*, Oct. 2021, pp. 1–8.

[85] P. Begunkov, "dlmCl: Optimization of CPU-GPU memory transfers for OpenCL devices with HSA," in *Proceedings of the 5th International Workshop on OpenCL*, ser. IWOCL '17. New York, NY, USA: Association for Computing Machinery, May 2017, pp. 1–2.

[86] N. Capodieci, R. Cavicchioli, and M. Bertogna, "Work-in-Progress: NVIDIA GPU Scheduling Details in Virtualized Environments," in *2018 International Conference on Embedded Software (EMSOFT)*, Sep. 2018, pp. 1–3.