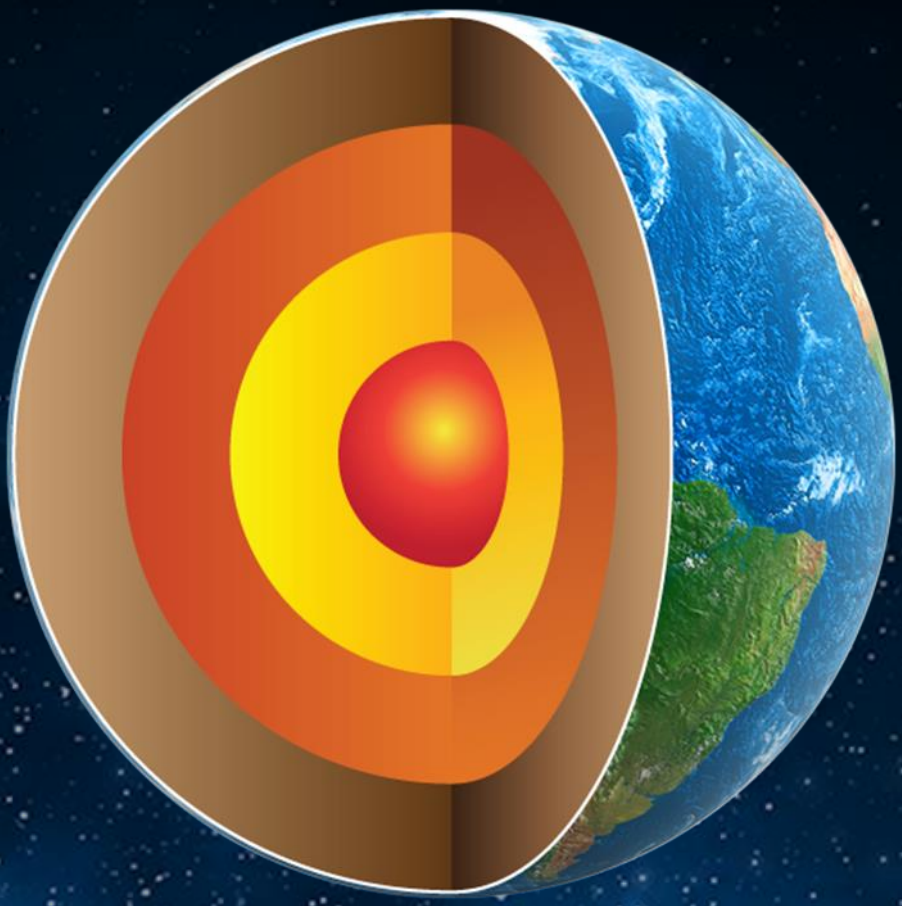
AMD



WHITE PAPER

MANTLE

EMPOWERING 3D GRAPHICS INNOVATION

TABLE OF CONTENTS

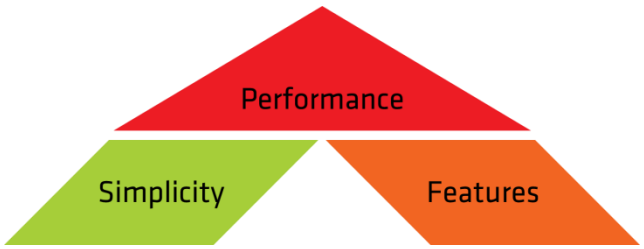
INTRODUCTION

Mantle is an initiative from AMD to create a new programming model that can fully exploit the capabilities of modern GPUs. In collaboration with leading cross-platform game developers, AMD has created an API specification and graphics driver that enable this model on PCs with graphics hardware based on the Graphics Core Next (GCN) architecture. Mantle is all about removing the obstacles that constrain game developers today and allowing them to unleash the full power of modern GPUs.

This white paper describes the motivation behind the creation of Mantle and provides a high-level overview of its most important features and benefits.

DESIGN PHILOSOPHY

The Mantle API has been designed to provide just the right level of abstraction for GPU hardware, striking a better balance between lower level control and higher level ease-of-use than existing programming models. At its foundation, Mantle strives to achieve a balance between simplicity and feature enablement, carefully selected to improve graphics rendering performance.



Simplicity does not necessarily mean that important features need to be sacrificed, but rather that the API should be easy to understand and provide predictable results. The goal is to avoid sources of complexity that don't provide any useful functionality in return.

Key Concepts

This design philosophy was translated into the fundamental concepts that underlie the Mantle API:

- **Pre-building and re-use of data**

  It's more efficient to do something just once ahead of time and re-use the results than to repeat that same process over and over again. Mantle utilizes that approach and encourages developers to do the same in their applications.

- **Control over memory management**

  Mantle lets applications manage GPU memory directly, providing opportunities to lower the memory footprint of their applications and significantly reduce driver overhead.

- **Control over command generation and execution**

  Mantle allows applications to directly control GPU execution in both single- and multi-threaded environments. This makes it possible to optimize performance, reduce latency, deliver more consistent frame rates, and more.

These concepts are applicable to a range of modern GPU architectures, and are not specifically tied to GCN. Mantle is intended to provide a thin hardware abstraction layer that can be broadly compatible with both current and future architectures, while allowing architecture-specific and platform-specific features to be exposed through an extension mechanism.

Why a New API?

While Mantle is far from the first attempt to create an API that achieves these goals, there are a number of reasons why AMD took the approach of designing something new instead of adapting or modifying existing APIs:

- **Success of the GCN architecture**

  Graphics Core Next is the first GPU architecture that provides a unified development platform across game console, PC gaming, and professional graphics ecosystems. This makes it a great baseline on which to establish a new programming model.

- **Developer demand**

  Mantle was designed in direct response to requests from developers of games and other applications who felt limited by existing graphics APIs. These developers did not believe that tweaks to existing programming models would be sufficient to solve the problems they were facing, but rather that more fundamental changes were required.

- **Performance and control**

  Getting the most out of today's hardware necessitated a fresh look at how existing programming interfaces were designed, along with a careful re-evaluation of what functionality was truly necessary. In some cases, maintenance of legacy API features that were no longer used by modern applications resulted in increased complexity and sapped performance. In other cases, abstraction layers designed to accommodate older and less functional hardware ended up placing unnecessary constraints on developers and limited their ability to control the GPU directly.

- **Excitement and innovation**

  Encouraging innovation means getting the development community excited about something new and different, and applying a series tweaks and patches to existing programming models isn't always the best way to capture attention. Mantle aims to kick-start a new round of 3D design and experimentation by offering a fresh approach to GPU programming, a wide array of new features, and unprecedented performance.

BENEFITS OF MANTLE

Mantle is designed to address real-world problems based on the guidance of game developers. This offers some compelling benefits to both developers and end-users, including:

- **Empowering lower spec systems**

  Mantle allows cutting edge graphics capabilities to be enabled on less powerful systems than ever before. This expands the range of devices that a developer can target with their applications, and reduces the number of image quality trade-offs they need to make. As low power devices continue to gain in popularity and get more attention from developers of graphics-intensive applications, the need for a more efficient programming model like Mantle will only increase.

- **Delivering more predictable performance and behavior**

  Mantle provides developers of PC applications with unprecedented control over the GPU through a very thin and efficient driver layer. This makes it uniquely positioned to deliver the best gaming experiences by enabling better frame rate consistency, in addition to higher throughput.

- **Sharing optimizations between PCs and game consoles**

  With the GCN architecture powering the new generation of game consoles and a significant portion of the PC market, Mantle finally provides a convenient path for developers to share optimizations and advanced rendering techniques between consoles and PCs.

By giving developers more direct access to PC graphics hardware, Mantle shifts much of the burden for ensuring optimal performance and stability away from the graphics driver and into their hands. This level of control may not be suitable for everyone, though those who have experience programming for game consoles or designing cross-platform game engines will already be accustomed to this level of additional responsibility. For those that care about maximizing performance, unlocking the full feature set of modern GPUs, or easily porting apps between consoles and PCs, Mantle is an ideal solution.

Figure 1: As the first title to support Mantle, Battlefield 4 from EA DICE unlocks new levels of performance for PC gamers, with stunning graphics to match.

MAXIMIZING PERFORMANCE

Mantle is engineered to be the fastest graphics API ever designed. It achieves this by improving GPU efficiency, reducing CPU overhead, providing more direct access to GPU hardware, and introducing an array of new performance-oriented features.

The Small Batch Problem

Mantle finally provides a solution to an issue that has bedeviled game developers for years, known as the "small batch problem." This refers to the situation where an application's performance becomes CPU-limited when a certain number of draw calls are generated per frame, leaving the GPU under-utilized. Developers have historically tried to deal with this problem by rendering larger batches of objects in a single draw call, which adds complexity to their code and places constraints on the artists responsible for developing game content.

With today's PC CPUs and industry standard APIs, most recent games can average 3,000-5,000 draw calls or objects per frame before hitting this limit. With careful optimization and high-end hardware, some developers have been able to get this number as high as 10,000 objects per frame.

With Mantle, the number of objects that can be drawn per frame can be increased by an order of magnitude, reaching 100,000 or even higher. This opens up many new possibilities for delivering richer and more compelling 3D graphics.

Figure 2: The Star Swarm demo from Oxide Games features massive space battles with tens of thousands of unique, dynamic 3D objects rendered at interactive frame rates. Based on the Nitrous game engine, it uses Mantle to dramatically reduce CPU draw call overhead.

Multi-threading

Reducing API overhead for draw calls still isn't enough to deliver all the performance improvements developers were asking for. Taking advantage of today's multi-core CPUs with effective multi-threading is also very important. CPUs with four, six, or eight cores are now commonplace, yet most of these cores are not accessible for driving GPUs. Mantle aims to unlock the potential of these multi-threaded processors and deliver near-linear scaling of draw call throughput as more cores become available.

Memory Usage

Efficient management of GPU memory is often a challenge on PCs. In contrast, past game consoles have been able to deliver great visuals using much less memory than PC graphics cards. With the latest game consoles including much larger amounts of graphics memory, this discrepancy puts more pressure on developers trying to maintain similar quality and performance levels in the PC versions of their games. By providing more control over memory allocations and more flexible access to the data they contain, Mantle finally makes these kinds of memory optimizations possible on PCs.

Direct GPU Control

Lack of direct GPU control sacrifices both CPU and GPU efficiency. It also makes latency more difficult to manage, and often results in unpredictable performance and behavior. Today's graphics drivers are very large and complex pieces of software, responsible for handling many things going on under the hood and hidden from developers. Mantle provides a more efficient path to translate API commands into the commands that GPUs understand.

Figure 3: Thief, a game developed by Eidos Montreal and Nixxes Software, takes advantage of Mantle to efficiently render its detailed and atmospheric 3D environments.

MANTLE CORE FEATURES

Execution Model

A fundamental feature of Mantle's operation is its execution model. A typical GPU has a number of different engines that execute in parallel – graphics, compute, DMA, multimedia, etc. The basic building block for GPU work is the command buffer, which contains rendering and compute commands targeting one of the GPU's engines. Command buffers are generated by driver software and added to an execution queue. When the GPU is ready, it pulls command buffers from the queue and begins to execute them.

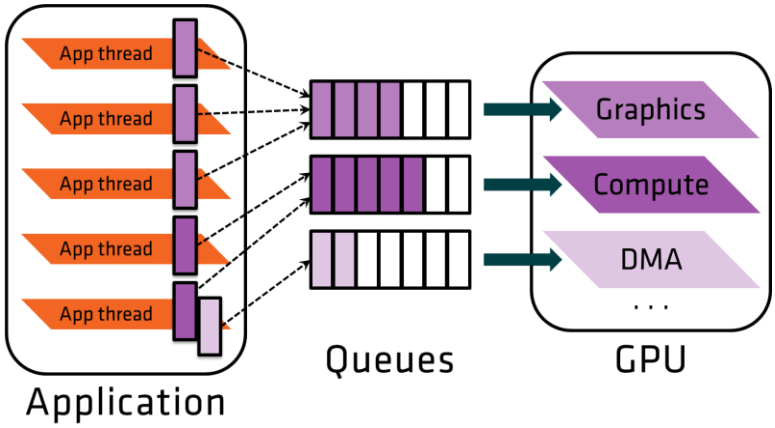


Figure 4: Mantle Execution Model

Mantle provides applications with a new level of control that allows them to make full use of the GPU's execution capabilities. It no longer leaves the graphics driver responsible for deciding which engine should execute which API commands, dividing work into command buffers, and managing synchronization between command queues. Most importantly, it enables multiple application threads to independently construct command buffers in parallel. This concept is fundamental to enabling very high API performance on modern CPUs.

Generalized Resources

Existing graphics APIs have a lot of semantics attached to resources (index buffers vs. constant buffers vs. vertex buffers, normal vs. staging resources, etc.). Many of these exist to provide hints to the graphics driver about preferred memory location for resources, or in some cases to allow compatibility with older GPU architectures. Mantle simplifies resource management and makes it more convenient.

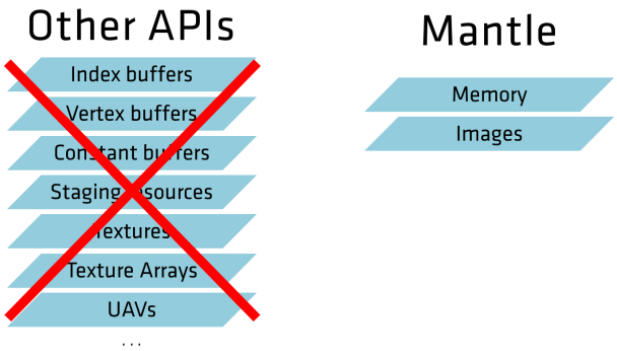


Figure 5: Generalized Resources

Memory Management

Mantle has been designed around the virtual memory capabilities present in modern GPU hardware. This provides a more general and flexible solution for memory management, and also provides a solid foundation for the future evolution of the PC platform.

In existing APIs, many data objects (such as "images" or "buffer objects") are directly tied to memory allocated by the graphics driver when they are created. This mechanism has several drawbacks, including the inability to efficiently recycle memory, a larger total memory footprint, expensive resource creation, and a general lack of efficiency stemming from the need to manage a large number of objects.

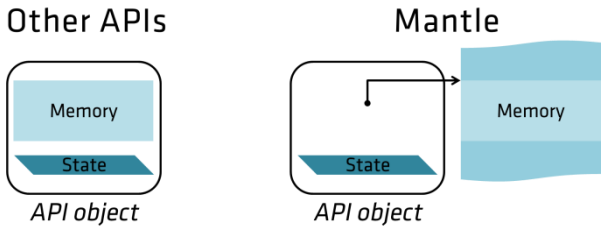


Figure 6: Decoupled GPU Memory

With Mantle, most objects are decoupled from GPU memory, and are instead treated as lightweight CPU-side data objects. This helps solve the problem of GPU memory management efficiency.

Explicit control of GPU memory gives developers the opportunity to better optimize usage of that limited resource. Mantle gives developers the option of controlling CPU memory allocated by the graphics driver as well, which can be critical for ensuring efficient multi-threaded operation.

Another interesting new feature of Mantle is the ability to re-map or "patch" process page tables for virtual GPU memory allocations. This functionality serves as the foundation for partially resident textures or tiled resources, and could be leveraged for many other interesting scenarios as well (such as flexible content streaming).

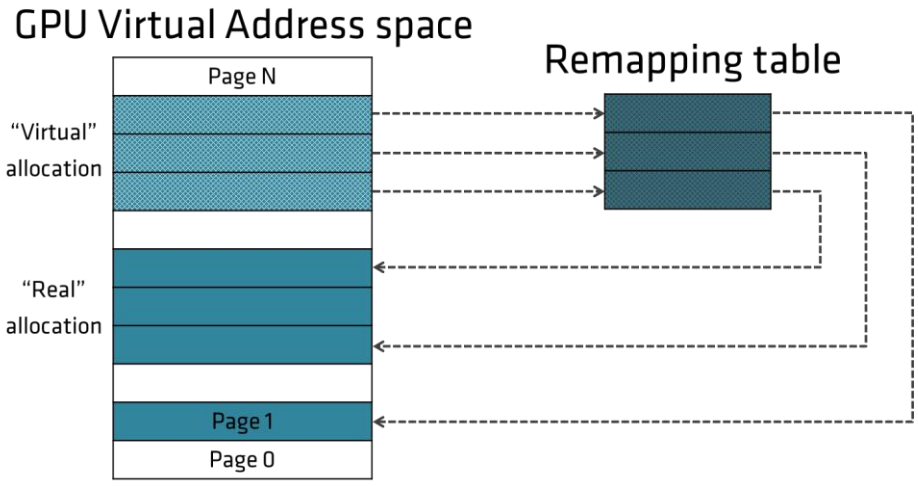


Figure 7: GPU Page Table Remapping

Monolithic Pipelines

Mantle introduces the concept of a monolithic pipeline object, which defines a large part of the state associated with a 3D pipeline using a single bind point. This includes all of the shaders in the pipeline, as well as certain fixed-function state that impacts shader execution.

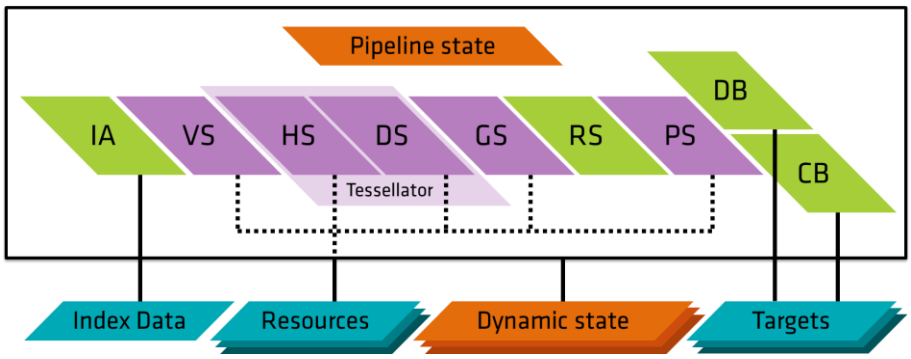


Figure 8: Graphics Pipeline Example

There are numerous benefits to this approach. The state and shader management model in existing APIs has proven to be a common CPU bottleneck in modern games, and Mantle's pipeline objects allow a reduction in API overhead by enabling up-front shader optimization at compile-time. They also make the CPU performance of the Mantle driver more predictable, since shader compilation will no longer get kicked off by the driver at draw time outside of the application's control.

Besides providing improved performance and predictability today, the pipeline abstraction creates a solid foundation for future development of Mantle. We expect a lot of innovation will be coming in that area in the future, enabling rendering techniques never before possible in real-time graphics.

Pipeline Saving and Loading

With Mantle, applications don't have to compile their shaders every time they are started up. Shader compilation is an expensive operation that can have a significant impact on load times, and can often be responsible for frame rate hitches if it takes place during gameplay. Mantle makes it possible to save and load complete pipelines with pre-compiled shaders quickly and easily, bypassing run-time shader compilation and avoiding unnecessary CPU overhead.

Resource Binding Model

Binding resources for shader pipeline access is another area where Mantle presents an improvement on existing programming models. The traditional resource binding model, where an application binds resources to fixed "slots" has created a CPU bottleneck in many modern games.

Alternative methods have been proposed, in particular a "bindless" model where resources are referenced from shaders via unique names or pointers instead of pre-defined slots. This method has some weaknesses of its own though, since tracking of the various pointers increases shader complexity and can result in behavior that is less GPU cache-friendly.

Mantle uses a new hybrid binding model that provides the best of both worlds. Descriptor sets can be created once and then re-used across multiple shaders and/or pipelines, which can often remove a substantial amount of CPU overhead. It is also possible to change specific resources within a set without having to re-bind the whole set, or bind pointers to resource subsets that can be updated independently. These capabilities enable fast resource updates and simplify resource management.

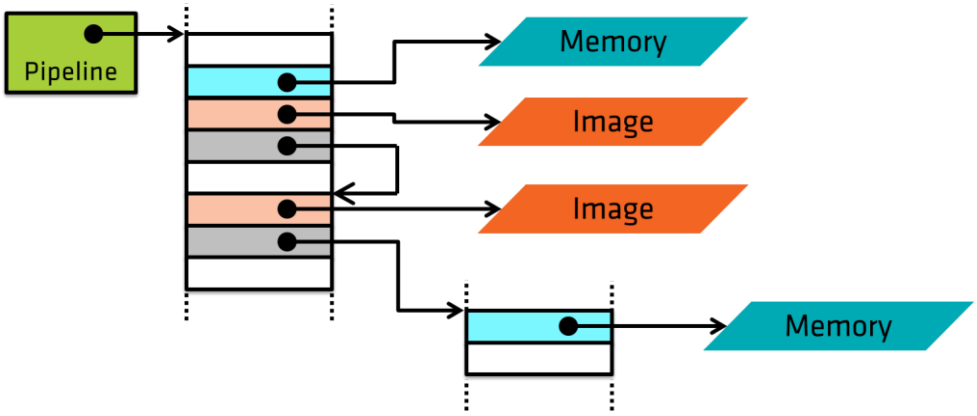


Figure 9: Mantle Resource Binding Model

Resource Preparation

A major source of driver overhead and unpredictability in existing graphics APIs is resource tracking. Graphics drivers need to understand resource usage to ensure that writes to a resource have completed before reads start from that resource, so that race conditions and data hazards can be avoided. They might need to flush GPU caches, decompress images, or perform other operations as well, all outside of the application's control.

This tracking is expensive in terms of CPU time, and in most cases redundant since most multi-platform game engines already do this on their own. Applications are in a much better position to handle this anyway, since the developer has an understanding of the complete rendering process and what stages are being executed at any given time.

With Mantle, the driver performs no automatic resource tracking. Instead, the application is required to explicitly report state changes to the Mantle driver. The performance costs of these actions are made very explicit, allowing developers to avoid unexpected frame rate drops and hitches.

As an example, when drawing to a color render target, the GPU may use some form of compression to reduce memory bandwidth. It may also leave behind portions of the image's pixels in some of the GPU's caches. If another shader wants to read the rendered image, it first needs to wait for the rendering of that image to be fully completed. It may also need to wait for the GPU to decompress the image into a native format that can be understood by the shader, and it may be necessary to flush out some cache lines to ensure they do not contain any leftover data from previous operations.

With existing APIs, all of these considerations would need to be handled by a graphics driver that must make assumptions about when and if they needed to occur. With Mantle, this process would be handled by an application through an explicit state transition operation. Since the developer explicitly controls when that occurs, they can optimize resource preparation to enable efficient rendering of large numbers of objects.

MANTLE EXTENDED FEATURES

In keeping with the philosophy of encouraging innovation, the Mantle API has been designed to support feature extensions that go beyond the core functionality. Some of the extended features provided with the initial release of Mantle are described here.

Advanced Anti-Aliasing Functions

The GCN architecture supports a range of advanced multi-sample anti-aliasing controls that allow games to enhance image quality and performance. Mantle includes extensions that expose these capabilities, enabling programmable sample patterns, independent control over color/coverage/depth samples per pixel, and direct shader access to MSAA surfaces.

Advanced Flow Control

Mantle includes innovative features that provide control over task scheduling and execution, enabling the implementation of algorithms previously impossible on GPUs.

Command buffers can be constructed with multiple monolithic and/or compute pipelines and conditional execution. Dynamic control flow between operations within a command buffer can be handled by the GPU without CPU intervention. This includes loops and "if-then-else" conditional statements, which can be nested to simulate case statements and limited recursion.

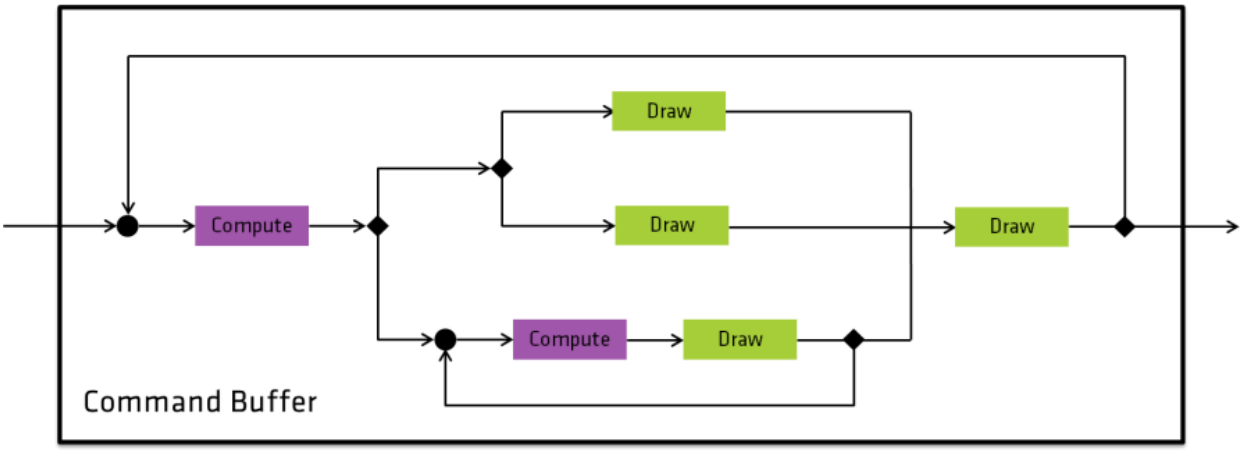


Figure 10: Mantle Command Buffer Example with Advanced Flow Control

Multi-Device Support

Existing API's provide limited support for multi-device rendering, which aims to scale up performance by splitting the workload across two or more GPUs. AMD CrossFire and other similar technologies allow this workload splitting to take place in the graphics driver without giving explicit control to the application or the graphics API. They typically do this by assigning alternating frames to each GPU for rendering, then compositing the outputs before sending them to a display device.

Mantle treats multi-GPU operation as a natural extension of the native single GPU execution model, scaled up to support multiple devices. Applications can control all of the GPUs and engine queues in a system in fundamentally the same way as they do for a single GPU. Mantle also includes support for hardware compositing, efficient data sharing or transfers between GPUs, and other features required to enable efficient multi-GPU solutions.

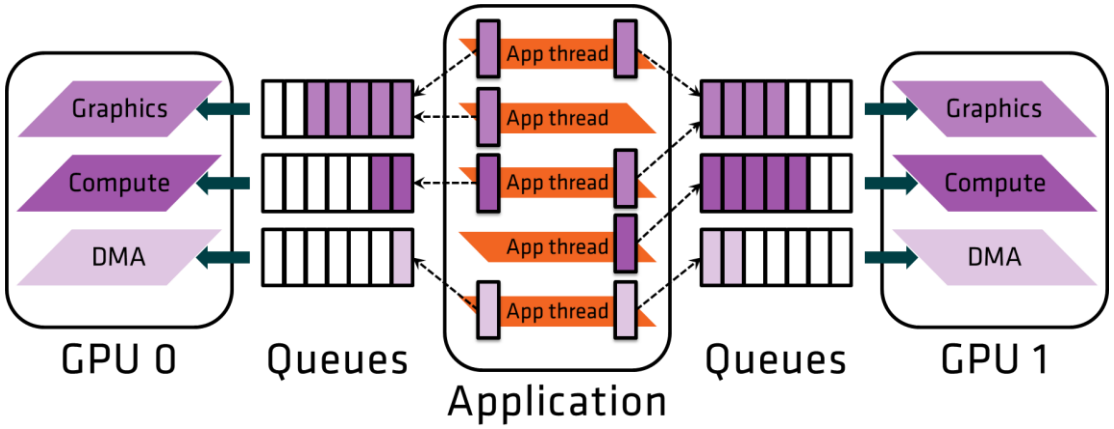


Figure 11: Mantle Multi-Device Execution Model

This multi-device execution model provides opportunities to achieve better performance scaling than is possible with current multi-GPU technologies that rely on alternate frame rendering techniques, along with much better control of latency and frame pacing. This is especially important for modern games that use more GPU compute functionality and other complex processing with frame-to-frame dependence. Mantle also enables games to take full advantage of asymmetric GPU configurations, including combinations of integrated APU graphics with one or more discrete GPUs.

DEBUGGING AND DEVELOPMENT TOOLS

Effective tools are very important to developers, particularly when dealing with an API that provides low-level access to GPU hardware. To this end, many debugging and validation features have been built directly into the Mantle API and driver. The extensive infrastructure includes many controls to make it less intrusive and performance-impacting when it's not needed. This means that in many cases, building tools simply involves building a user interface on top of capabilities already included in the API. This all translates into a better development experience.

CONCLUSION

Mantle offers new and exciting ways to unleash the full capabilities of the latest GPUs. It was designed to improve today's graphics applications by providing higher frame rates and more consistent performance, and also to pave the way for future industry standards. It will benefit both the most powerful hardware by enabling new levels of performance and graphical fidelity, as well as low power hardware by eliminating unnecessary processing overhead. And it opens the door for developing entirely new kinds of 3D rendering techniques that previously existed only in the dreams of developers.