



White Paper

RAY TRACING FOR THE MASSES

Creating immersive, energy-efficient graphics experiences with the Photon architecture

By Kristof Beets, Vice President of Technology Insights, Imagination

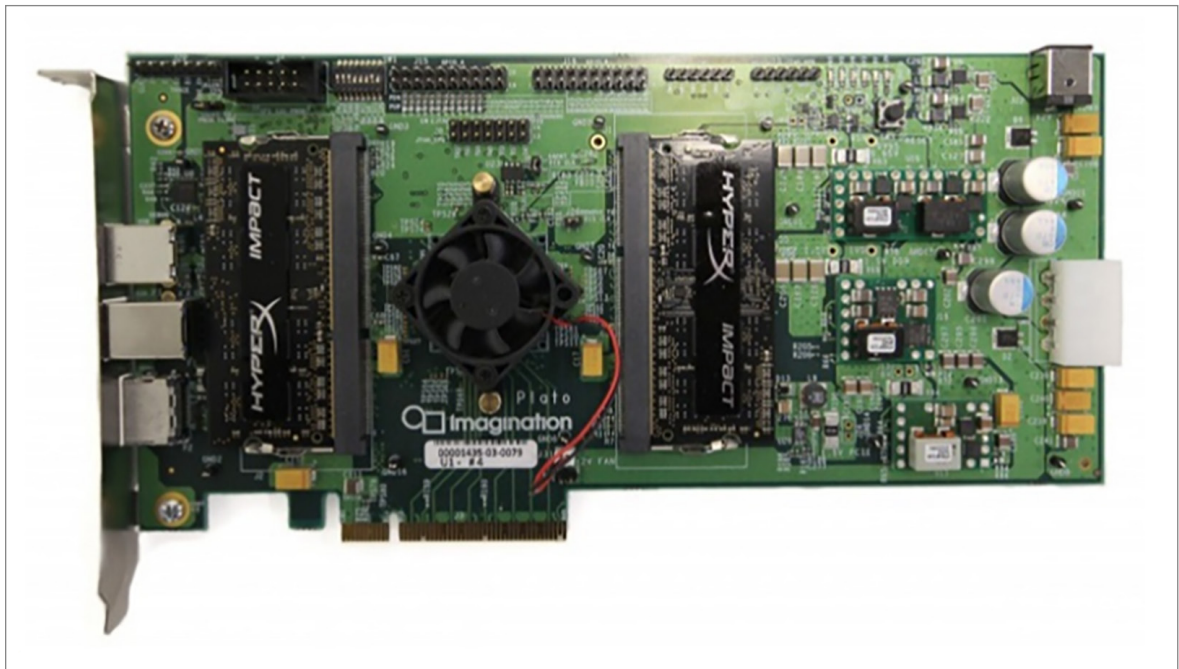
January 2023 update - IMG DXT

Intro

In 2021 Imagination introduced its PowerVR Photon Architecture which enabled real-time ray tracing within its CXT GPU family. Often described as the 'holy grail' of computer graphics, ray tracing is where a 3D scene is generated using a technique that mimics how light behaves in the real world, thus providing developers with the tools to make incredibly realistic visuals.

Now in 2023 Imagination is introducing an evolution of its PowerVR Photon architecture-based Ray Acceleration Cluster (RAC) into its IMG DXT graphics processing units (GPUs).

The focus for 2023's DXT is on improving efficiency through the introduction of Fragment Shading Rate (FSR) functionality and enhanced post-processing techniques while changes to the layout deliver scalability and flexibility for ray tracing.

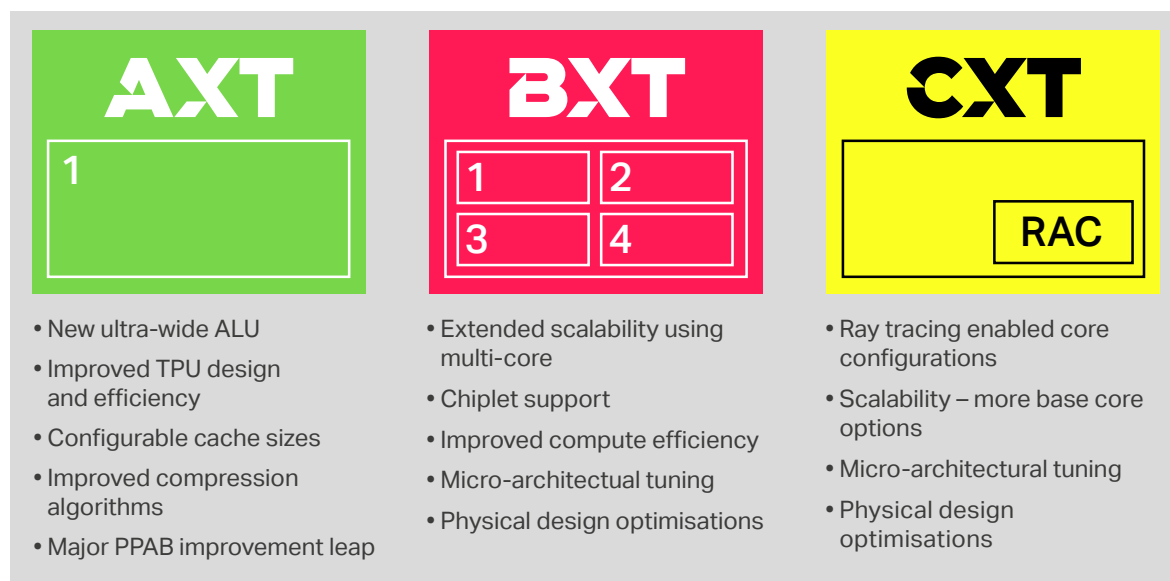


Imagination first announced its ray tracing technology in 2014, and in 2016 produced a demo board known as "Plato".

PowerVR GPU Architecture

Imagination has been delivering market-leading efficiency-focused GPUs for over 25 years and the IMG DXT is the latest incarnation of this architecture. It builds on years of innovation and differentiation with a strong focus on power and bandwidth efficiency, which have always been the cornerstones of the design principles at the heart of Imagination's IP.

Evolution



While PowerVR GPUs have evolved over many generations since 1996, here we focus only on the latest iteration of our architecture, which started with the IMG AXT series in 2019. AXT was a radical redesign, making a significant leap forward in power, performance, area and bandwidth (PPAB) efficiency. This was achieved through a revolutionary new ultra-wide arithmetic logic unit (ALU), which processed up to 128 scalar data lanes concurrently within a task (warp). To ensure the highest utilisation of this ultra-wide architecture many novel scheduling advances were introduced, including triangle merging and task packing.

To keep bandwidth low, we also introduced the latest version of our PVRIC compression, which added lossy framebuffer (render target) compression to our already extensive usage of lossless compression, for depth buffers, render targets, textures and geometry. The texture unit was also completely redesigned for quality and bandwidth efficiency and a customer-configurable cache architecture was enabled.

In 2020, the IMG BXT series added another innovation: decentralised multi-core. This enabled scaling beyond our traditional smartphone market and opened up new business opportunities in the desktop, data centre and automotive ADAS market segments. This multi-core approach fully embraces new silicon process nodes with a strong focus on physical design optimisation and tuning for the best PPAB as well as new technology directions, such as chiplets, that bring cost and efficiency benefits. Process-node-specific tuning is increasingly important for GPU design, to deliver the most optimal sweet spot designs for 7nm, 5nm and even 4nm process technology.

In November of 2021, we revealed the IMG CXT series and its standout feature: the PowerVR Photon architecture, offering hyper-efficient hybrid ray tracing.

IMG DXT – what is new?

Where the C-Series had a strong focus on ray tracing the design investment for D-Series is wider. It offers improved scalability and granularity (both for ray tracing and traditional rasterisation), improved efficiency and utilisation through enhanced scheduling and texturing, and a feature focus. The following sections will dive into each improvement in turn.

Scalable Ray Tracing

While ray tracing is a highly desirable feature offering numerous visual quality and efficiency benefits the fact remains that it is new, especially in the mobile market, and hence the ecosystem of content is in its infancy. The result of this is that OEMs question if and how much they would like to invest in this new technology. The ray tracing silicon logic offers little benefit when it is not used, which is true for all levels of ray tracing acceleration – since even the Level 2 ([see our "What's Your Level?" white paper](#)) box and triangle testers become dark silicon when running traditional graphics content.

As a result of the above market reality Imagination has been looking to offer more flexibility to its customers by scaling the amount of ray tracing offered in its GPUs. Of course, the Ray Acceleration Cluster (RAC) units are optional and can be fully removed if desired, enabling us to offer GPUs where all the investment in silicon area is in traditional rasterisation. Additionally, we have been investing in wider scalability of both performance and silicon area budget where instead of just offering a single RAC configuration we now offer a wider range, with either full or half RACs.

Finally, we also enable a wider range of ratios between traditional processing and RAC units. Where previously the RAC was shared between two ALU blocks, in this generation, we offer more flexibility by enabling the RAC to be shared by many more ALU blocks and moving the RAC to different levels with the GPU hierarchy. Both these elements work in tandem with the following architectural change.

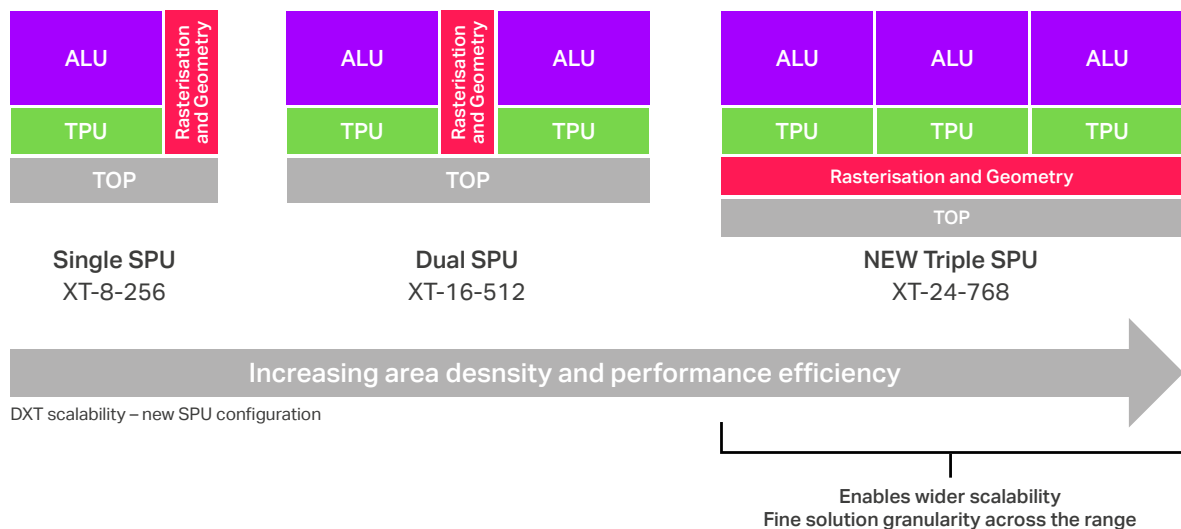
As we look to enable GPU cores with less ray tracing, we have also continued to invest in post-processing technology to help make designs with low-ray budgets viable. This has included investment in faster and more efficient post-processing for features such as denoising and super-resolution and technologies such as Fragment Shading Rate, which trades quality for performance. All these complimentary improvements, which enable the efficient use of reduced ray budgets, will be highlighted in this white paper.

GPU Scalability and Granularity

When the PowerVR Rogue architecture was introduced in 2012 it featured a design with dual ALUs (inside Unified Shading Clusters or USCs) which shared a single Texture Processing Unit (TPU), similar to how the RAC was shared in the C-Series between the two ALU blocks. In this design, fixed function units and other shared processing resources inside the GPU were all focused on this balance point. The shared TPU concept was dropped in the A-Series, which instead offered a 1:1 ratio of ALU to TPU blocks. This combination of two USCs, two TPUs and the shared logic which included geometry and rasterisation logic, was dubbed a Scalable Processing Unit, or SPU.

IMG DXT – what is new?

With IMG DXT we are now for the first time evolving this base concept of scalability by allowing SPU's with more ALU/TPU processing performance. Specifically, we are introducing a new, higher-density SPU design, with three USC/TPU blocks. This change is illustrated below.



Note that since AXT we have also supported a special minimal-silicon-area GPU configuration featuring an SPU with a single ALU/TPU combination for cost-sensitive market segments.

The 2021 premium smartphone market sweet spot for 4/5nm designs – CXT-48-1536 – was constructed as a three-SPU design and within that generation could be combined with one RAC per SPU each: hence a CXTP-48-1536 RT3 configuration.

With the IMG DXT, we can build many more options, since in addition to the three SPU's, (which is effectively three 2x 8-256 SPU's), we can now build this as two SPU's but with three USC/TPU units (making it effectively two 3x 8-256 SPU). These can now be combined with a RAC each, creating either a DXT-48-1536 RT2, or with a half RAC each, delivering a DXT-48-1536-0.5RT2. This shows that for the same performance point in terms of floating point and texturing, we can now offer RT1, RT2 and RT3 configurations with high efficiency and, as well as an RT4 with our largest possible single core.

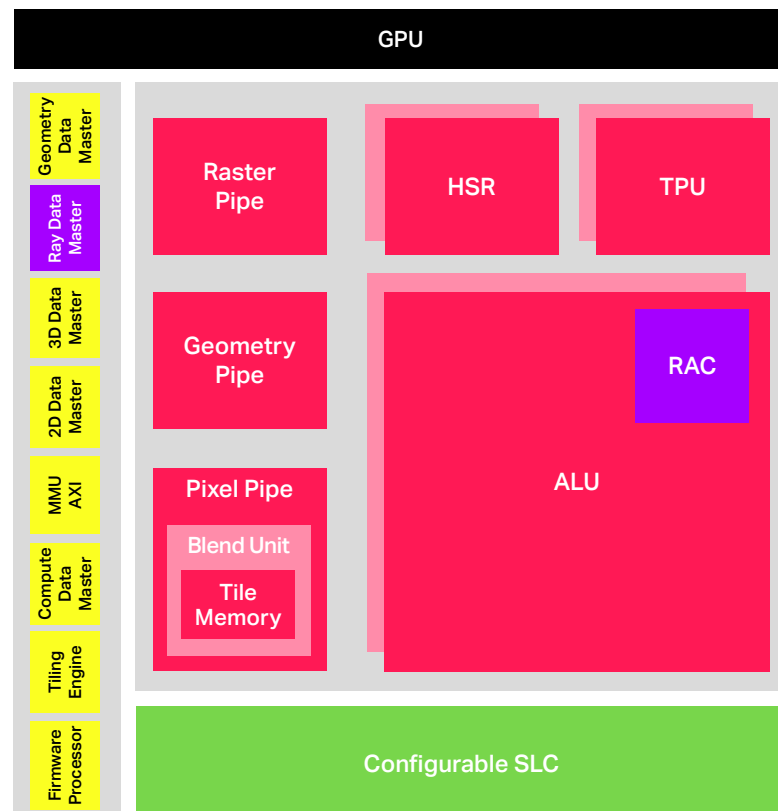
While the SPU's are larger individually they consume less silicon area overall, helping to improve density. They also allow for a higher peak configuration since within single-core designs we still allow up to four SPU's. This means the largest possible single-core configuration is now upgraded from a CXT-64-2048 to a DXT-96-3072, which effectively is 50% higher single-core performance compared to the previous generation. The combination of both two USC/TPU SPU's and three USC/TPU SPU's in our designs allows us to expand the range of cores we can deliver to customers as is shown in the table below.

IMG DXT – what is new?

Configuration	CXT	DXT	DXT ray tracing options
8-256	Y	Y	½ or 1 RAC
16-512	Y	Y	½ or 1 RAC
24-768	Y	Y	½ or 1 RAC
32-1024	Y	Y	1 or 2 RACs
48-1536	Y (three SPUs)	Y (two or three SPUs)	1, 2 or 3 RACs
64-2048	Y	Y	2 or 4 RACs
72-2304	N	Y	1.5 or 3 RACs
96-3072	N	Y	2 or 4 RACs

Pipelined Data Masters

PowerVR GPUs have been driven by a firmware processor since the very beginning. The firmware processor is responsible for high-level scheduling and priority of workloads and does so in combination with fixed function units: the data masters. To allow concurrent processing of different types of jobs PowerVR GPUs have a data master per job type, including geometry, 3D, compute and 2D (or data movement).

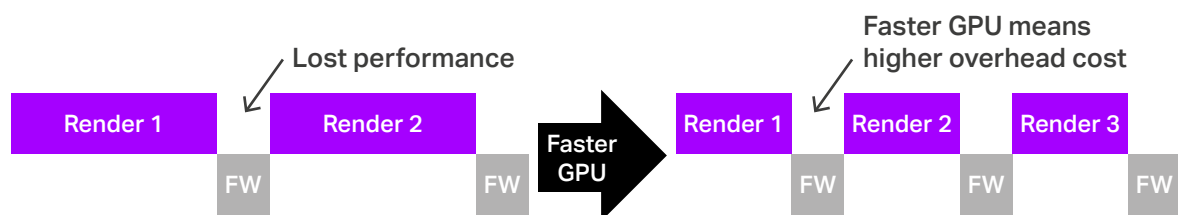


IMG DXT – what is new?

These data masters are responsible for the low-level running of these jobs, including setup work. Previous generations had single-tasking data masters, meaning that the data master would be executing a specific job and changing the job would require work by the firmware processor to set up the next job. This approach means that most of the setup work occurs when changing from one render to the next which would often lead to idle time during which the firmware processor is setting up the next job and reprogramming registers. This setup work may require data access and other complex synchronisation tasks, which due to latency could result in 1000s of cycles of firmware work during which no work is scheduled for the specific data master. This would often lead to idle time or even power-gating of the GPU core and thus lost performance and reduced scaling efficiency.

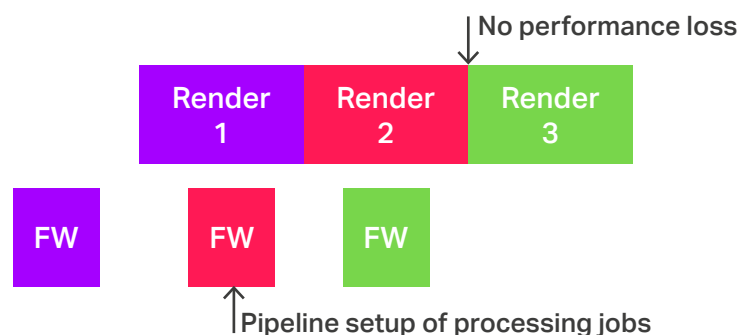
As our GPUs have become ever faster with more powerful SPUs, higher SPU counts and multi-core, the rendering/processing performance of the GPU has gone up significantly. This means that the time to process compute kernels and/or graphical renders reduces (as we have a larger, faster GPU core), but, we still have a single firmware processor which means the setup time stays the same.

As an example, comparing AXT-16-512 to DXT-72-2304, our processing is theoretically 4.5x faster, but the firmware processing time has stayed the same, hence it now represents a larger proportion of the total time. This is illustrated below:



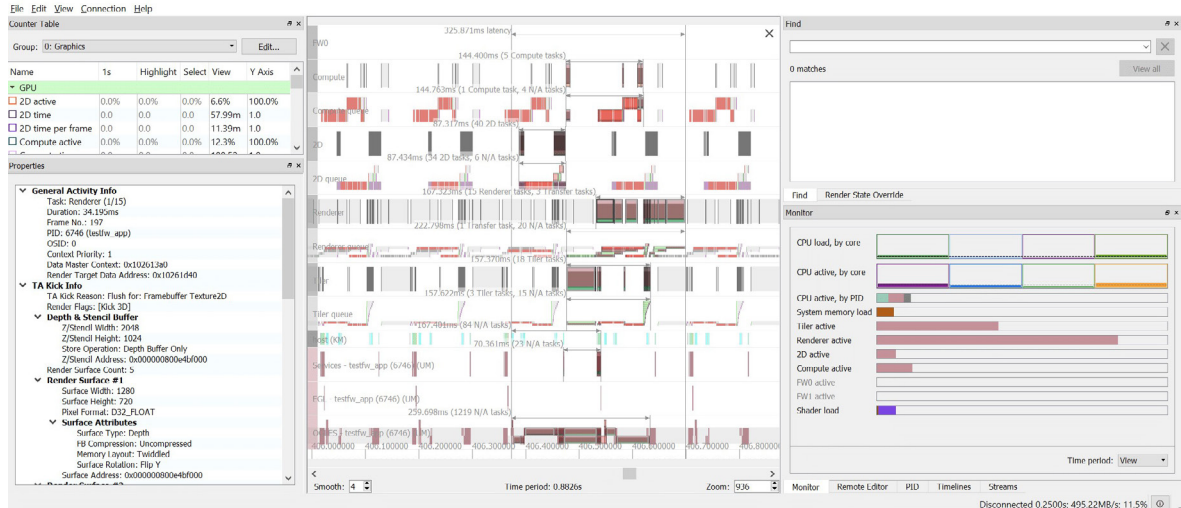
IMG DXT addresses this setup-time issue by introducing pipelining to the data masters. Pipelining means that the firmware can set up (pipeline) the next job while a previous job is still processing within the GPU. Effectively, the firmware work is now overlapped with GPU work instead of running serialised in-between jobs. This approach enables higher performance for the same level of core, as we avoid idle cycles and improve utilisation of the GPU processing hardware, which means a better return on investment.

The concept is illustrated below.



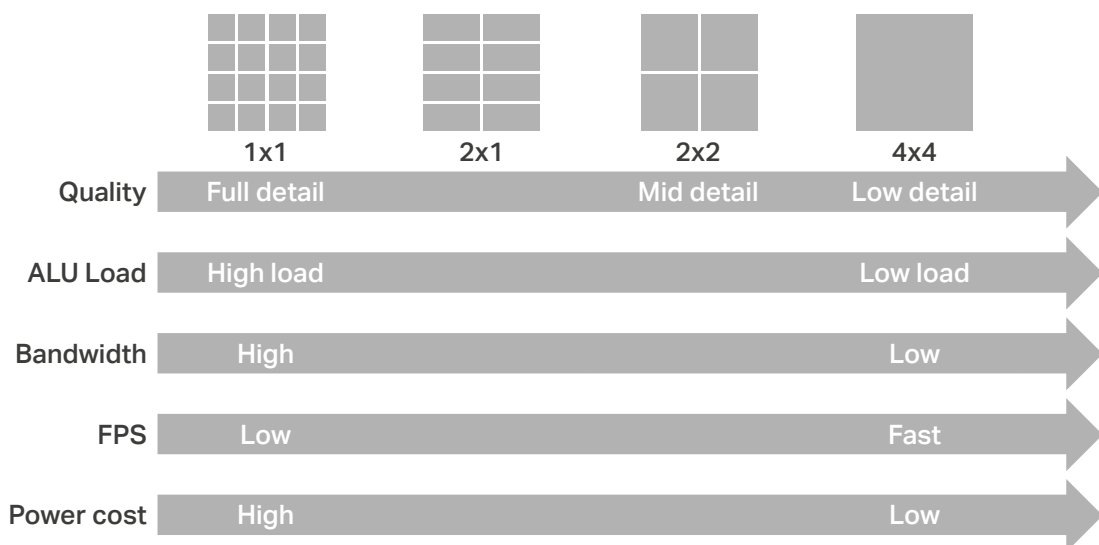
IMG DXT – what is new?

The idle time in previous generations could also be seen using our PVRTune profiling tool.



Fragment Shading Rate

Fragment Shading Rate or FSR (also known in DirectX as Variable Rate Shading), is a new technique for developers to trade image quality for improved performance and reduced power consumption. Specifically, FSR allows us to share the result of the execution of a shader program across multiple fragments. So rather than a 1:1 execution of a shader load program, which means one shader invocation per pixel/fragment, we can reduce this execution rate by executing the program once but allowing it to cover a region of two or more pixels. As we now execute less work it means higher performance, lower bandwidth, and lower power consumption.

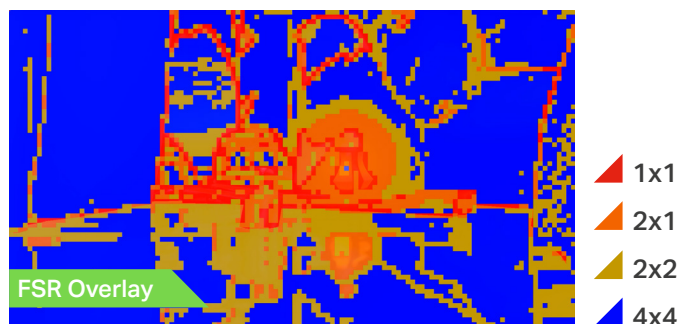
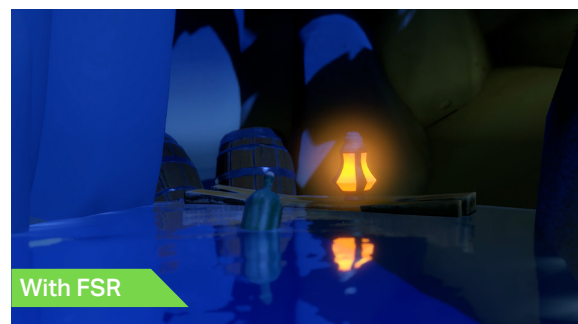
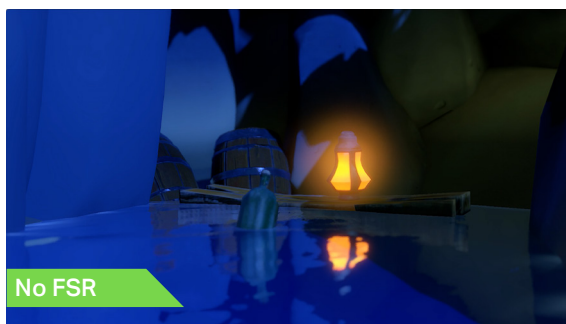


IMG DXT supports a wide range of ratios by supporting shader execution for the following zone sizes: 1x1 (as per normal), 2x1, 1x2, 2x2, 4x1, 1x4, 4x2, 2x4 and 4x4 (lowest quality, lowest execution rate). The Fragment Shading Rate is controlled either at the geometry/draw-call level where it can be set per draw call or even per primitive within a draw call. Alternatively, the rate can also be controlled by an image map, which is loaded or dynamically generated by the GPU.

IMG DXT – what is new?

Typically, the aim is to drop quality where it is less noticeable for the user, for example, in a racing game where trackside objects can be drawn at lower quality as they will be blurred out by the motion post-processing effects. In this scenario, the game engine can use the Vulkan® function calls to set up those draw calls to re-use the shader execution for multiple pixels thus offering a speed increase but with minimal impact on the quality. Similarly, a more dynamic approach can be used where based on a previous frame a contrast analysis can be done; for regions with little contrast and detail the pixel zone can be set to a reduced fragment shading rate thus reducing work and improving performance and power efficiency in the next frame.

The Fragment Shading Rate feature also works very well with ray tracing. As there are fewer shader invocations fewer rays need to be emitted and therefore, fewer rays are submitted for processing and the results are re-used over a larger zone of pixels. This approach is shown in the following screenshots from our D-Series launch demo. This shows the benefit of ray tracing in combination with Fragment Shading Rate for a significant cost reduction (shader processing, ray counts) with a minimal impact on perceived quality but offering large gains in performance and a reduction in power consumption.



2D Dual-Rate Texturing

For every GPU generation the performance teams within Imagination run through a wide range of content, analysing and understanding the different workload types and their bottlenecks. As part of this analysis, the data revealed that many modern games spend an increasing amount of time executing post-processing algorithms to enable depth of field, bloom and blur effects. Most of these post-processing passes are texture-sampling heavy filter effects which are modest in ALU requirements and hence bottlenecked by the throughput rate of the Texture Processing Unit (TPU). One approach to resolve this would be to simply brute-force change the ratio of the number of TPU units versus the USC/ALU rate. However, our analysis indicated this was not a good strategy, for several reasons.

IMG DXT – what is new?

First, in regular render passes the ratio of ALU versus TPU was already optimal and adding more TPU would simply not result in any benefits as the workload would become ALU limited. Meanwhile, other processing passes were TPU-heavy but also bandwidth-heavy, and hence boosting the TPU would not help, as there would be insufficient bandwidth to feed the extra TPU throughput so performance would not be enhanced. Our teams found that post-processing workloads as well as compute image processing workloads showed the following characteristics:

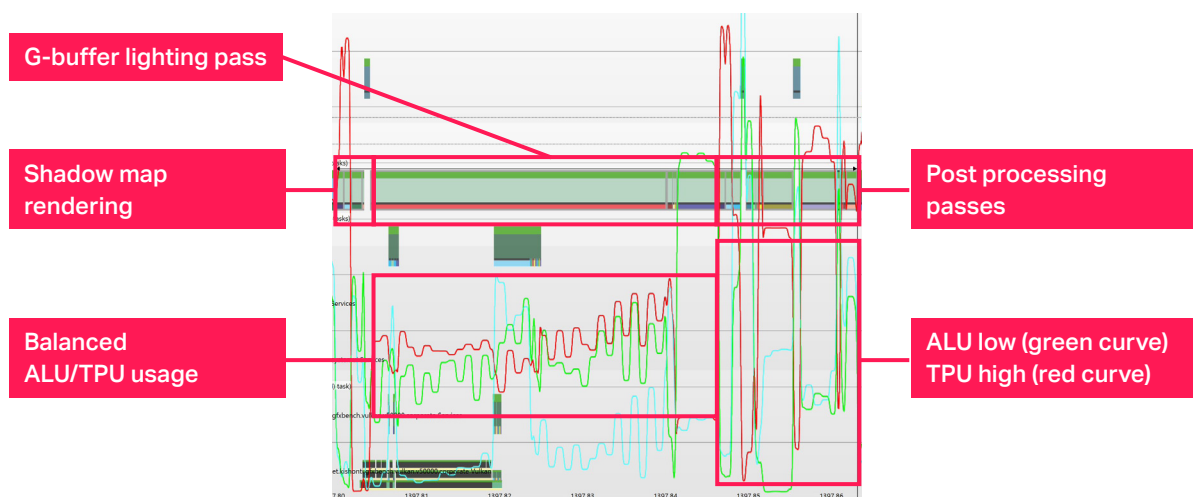
- regular processing/sampling across a region, with a large amount of re-use of sampling points which hit on the texture cache;
- 2D sampling of a single render target/texture with no LOD and no perspective.

The above two characteristics led us to implement a new TPU mode which allows the performance to be doubled up but only when the hardware detects these specific characteristics. The first characteristic is important as the regular sampling with high sample reuse (e.g., moving window filters) avoids bandwidth limits. The second is important as it allows us to keep the amount of duplicated logic low, hence offering doubled peak throughput rate but avoiding doubling all TPU logic. The result of this approach is a modest increase in TPU size, but doubles performance where it makes sense while remaining in balance with overall characteristics e.g.; we deliver a true speed up and avoid the ALU and/or bandwidth bottleneck cases where the TPU was already fast enough.

What this means is that for certain processing types, the DXT-48-1536 will effectively behave like a 96-1536, processing twice the number of bilinear-filtered texture samples per clock and hence delivering twice the execution rate versus the previous CXT-48-1536 generation.

As an example, the illustration below shows a typical mobile game with its render passes. The bar at the top, starting on the left, shows the various Vulkan® Render Passes, with several pre-processing passes which are typically for shadow maps, placing considerable pressure on the depth-test units.

The second phase of rendering is the main scene which in this case is a GBuffer render pass and a lighting pass. What we see is that this is the bulk of the processing time for the frame and the ALU and TPU loads are relatively balanced; this is illustrated by the curves in red (TPU load) and in green (ALU Load). We can see that over time both show average utilisation, which is typical for the main scene with a balanced mix of ALU and TPU work.



IMG DXT – what is new?

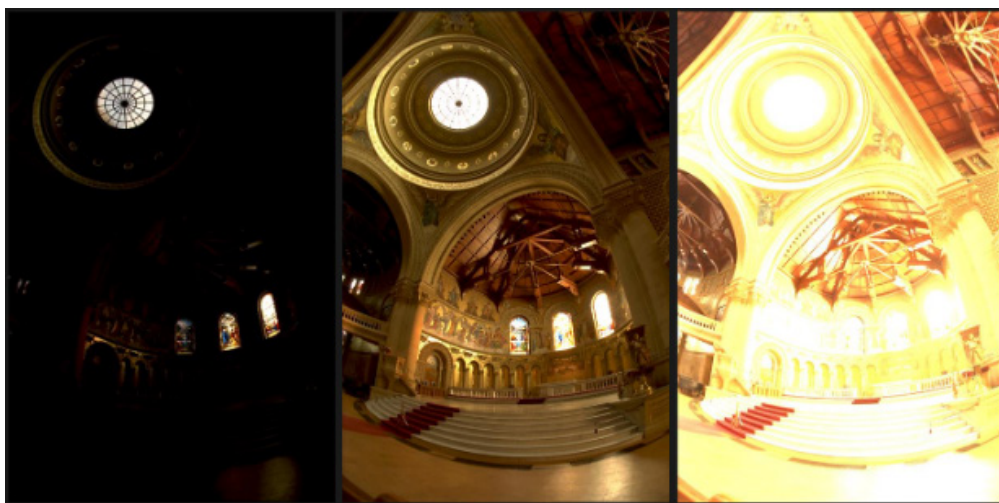
Most of the interest for us here is the last set of render passes, which are the post-processing passes. Typically, this is where bloom, blur and many other HDR-style post-processing effects are applied on top of the previous main render pass. What is notable here in that zone is that the red TPU curve shoots high for many of them, but the green ALU curve is very low. This indicates the TPU unit is causing a processing bottleneck – which is exactly what the 2D dual-rate TPU is designed to address. It doubled the speed of the TPU for these workloads, thus reducing the rendering time by a factor of two and speeding up frame rendering.

As already mentioned earlier in this whitepaper, one of the changes in DXT is to enable lower amounts of ray processing versus traditional processing, a change driven by the market ecosystem reality. With fewer rays per pixel available the quality and performance of the post-processing effects for denoising matters even more and, usefully, the 2D dual-rate TPU can help with this, and also for upscaling and super-resolution algorithms.

ASTC HDR Support

The Khronos Vulkan API mandates adaptive scalable texture compression (ASTC) low dynamic range (LDR) textures and PowerVR GPUs have supported these for several generations. However, the ASTC high dynamic range (HDR) mode has remained optional and rarely used. Discussions with developers and ecosystem influencers have made it clear that in the coming years, HDR will become expected for content and hence supporting HDR input textures compressed using the ASTC algorithms makes sense to add to our designs and with IMG DXT we fully enable this type of compression textures.

The benefit of HDR is that it allows textures to capture much more information encoding all the contrast from very dark to very light colours. HDR textures mean that you can see everything happening on screen the way it was intended, with true-to-life colours, shadows and detail – giving the ultimate visual experience. That visual range of information is illustrated below:



From Recovering High Dynamic Range Radiance Maps from Photographs by Paul E. Debevec and Jitendra Malik.

RISC-V Firmware Processor

Imagination's GPUs have long benefited from an integrated firmware processor, enabling many unique features such as:

- host CPU offload to handle GPU events directly;
- flexibility to enable new features and functionality as well as performance improvements;
- priority-based rendering using a wide range of algorithms enabled using the firmware code;
- error handling including functional safety;
- GPU kernel logs to help debug problems by dumping processing state and registers;
- GPU debug by enabling the step debugging and register access;
- hardware-based interaction with third-party blocks using GPIO signals from the processor.

While Imagination has used a wide range of processors over time, including MIPS and proprietary processors, with DXT we are switching the XT branch of our GPUs to a RISC-V-based firmware processor. Our automotive XS range of GPUs has been using RISC-V to enable functional safety and so as a silicon-proven certified approach this switch is low risk and benefits from RISC-V's latest advances in tool flows and C-programmable functionality. The RISC-V processor is also up to 40% faster than the previous proprietary processor, enabling even speedier event handling and more functionality.



Power Efficiency Optimisations

Power efficiency is at the forefront of all our improvements and many of the above architectural advances deliver more performance at the same or lower power consumption.

In addition, our performance analysis teams have been digging into the physical design characteristics of our IP, going through the full flow and measuring the power consumption by analysing the largest consumers of power under real practical relevant use. This insight has been used to help guide micro-architectural changes and improvements. One of the main consumers of power was found to be the USC (ALU) block and more specifically the register bank sizes and structures, and as a result of this analysis, these have been tuned to use SRAMs with lower power consumption and improved throughput.

PowerVR Foundational Features

The PowerVR architecture includes many patent-protected innovations, which for many years have offered unique benefits to our customers and partners, and many of these continue to offer value and these are highlighted in this section.

Tile-Based Deferred Rendering

The traditional rendering technique on most GPUs is known as immediate mode rendering (IMR), where geometry is sent to the GPU and gets drawn straight away. This simple architecture is inefficient, resulting in wasted processing power and memory bandwidth. Pixels are often still rendered despite never being visible on the screen, such as when a tree is completely obscured by a closer building.

PowerVR's tile-based deferred rendering (TBDR) architecture works in a much smarter way. It captures the whole scene before starting to render, so occluded pixels can be identified and rejected before they are processed. The hardware starts by splitting up the geometry data into small rectangular regions that will be processed as one image, which we call "tiles". Each tile is rasterised and processed separately, and, as the size of the tile is so small, all data can be kept inside very fast, yet low-power, on-chip memory.

Deferred rendering means that the architecture will defer all texturing and shading operations until all objects have been tested for visibility. The efficiency of PowerVR hidden surface removal (HSR) is high enough to allow overdraw to be removed entirely for completely opaque renders. This significantly reduces system memory bandwidth requirements, which in turn increases performance and reduces power requirements. This is a critical advantage for phones, tablets, and other devices where battery life makes all the difference.

PowerVR Image Compression

PowerVR Image Compression (PVRIC5) is the latest version of our framebuffer (render target) compression technology. Framebuffer compression aims to reduce the bandwidth and power consumption generated by reading and writing render targets to and from system memory. PVRIC includes a lossless compression mode; a variable compression rate mode where quality remains perfect, but the compression achieved depends on the contents of the render target. Typically, a reduction of 50% is obtained across a wide range of reference test images.

PVRIC also includes a visually lossless compression mode that guarantees a minimum compression rate, thus ensuring that memory footprint, bandwidth and power consumption are all reduced. PVRIC within DXT is focused on high quality with a wide range of data format support and offers a guaranteed 50% compression mode that offers visually lossless quality.

Ultra-Wide ALU

Since the A-Series, PowerVR GPUs have deployed a 128-wide arithmetic logic unit (ALU) architecture that embraces a simpler RISC-style form of ALU engine. By moving to simpler engines, we can pack many more lanes into the same silicon area and power budget, fully embracing massive thread-level parallelism and simplifying compiler effort to gain high utilisation and efficiency. This approach is further combined with a wide range of complimentary scheduling technologies which ensure this ultra-wide warp (task) size is used effectively, including triangle and task merging mechanisms.

PowerVR Foundational Features

This approach was critical to delivering the major improvement in density (fps/mm²) and power efficiency (fps/w) within the IMG A-Series, and DXT continues to build and refine this architecture

Superscalar ALU Processing

Imagination's ALU fully embraces superscalar processing, allowing many ALU types and co-processors to operate concurrently for the highest efficiency and utilisation. This includes floating-point pipelines, integer pipelines, complex ops pipelines, test/branch pipelines, bitwise operation pipelines and DMA units all working concurrently based on data availability and instruction mix in the shaders/kernels.

ASYNC Everything

Due to their tile-based deferred rendering (TBDR) nature, PowerVR GPUs inherently require asynchronous concurrent processing of different types of tasks. Traditionally this has been geometry (also known as "TA" which stands for tile accelerator) and pixel/fragment (also known as "3D") processing tasks running concurrently in the GPU. With the addition of compute (OpenCL) this was also enabled concurrently and today this is known as the "asynchronous compute" feature in most GPU specifications. PowerVR GPUs also support 2D/housekeeping tasks concurrently as special jobs for simple non-3D (geometry) processing as well as data movement. Since CXT, we now have concurrent asynchronous ray tracing added into the mix.

DXT GPUs maintain this capability and may now have up to five different task types executing concurrently within the GPU: geometry, fragment/pixel, compute, 2D and ray tracing.

Decentralised Multi-Core

The traditional approach to GPU scalability is limited by the connections between centralised shared blocks and the shader cores. Typically, the shared logic includes a centralised memory data path, job manager and geometry tiling engines. The central dependency generates a star-network-style structure where all cores need to be connected to this single centralised entity. However, this causes issues with congestion and layout flexibility.

Imagination's novel approach to multi-core instantiates a flexible number of GPU cores without a direct dependency on a connection to a central unit. In its simplest form, this can be seen as multiple GPUs, which are present in an SoC design, but with the ability for cores to jointly work on compute and graphics processing. Each core within this approach is designed as a fully independent GPU, containing all the functionality required to self-manage and execute workloads based on priority.

The fundamental change to previous layouts is that instead of a single core directly working on a workload we now have multiple GPU instances sharing command streams and jointly finishing the work as quickly as possible. By working on distinct regions of the work (the render target) in each GPU core our bandwidth efficiency is maintained, as each core continues to work on a coherent region of the screen, thus ensuring maximal cache hit efficiency. A similar approach also applies to other processing types, including geometry and compute processing, which can be assigned to different GPU cores for processing.

PowerVR Foundational Features

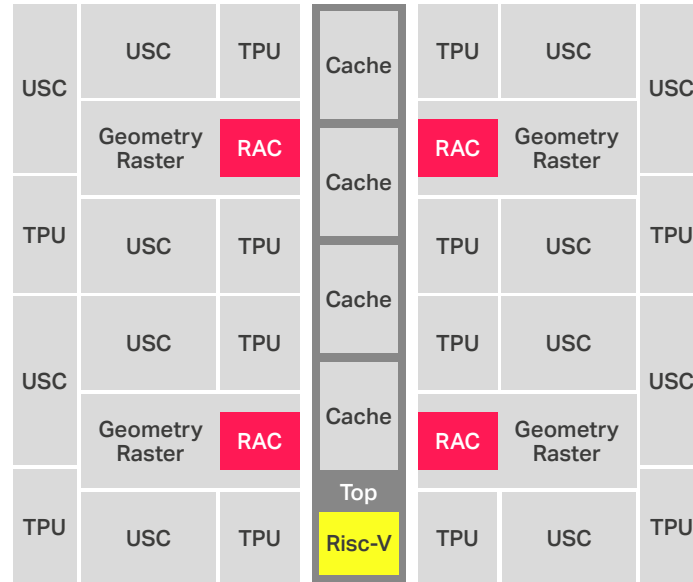
PowerVR's decentralised approach is a great fit for emerging physical design concepts, including chiplets, where a distributed working model offers numerous efficiency and dynamic flexibility benefits.

Automotive Grade

Thanks to the coherency gathering in the Level 4 Ray Tracing Levels System (RTLS) architecture (explained below), the IMG CXT and IMG DXT are also ideally suited for ultra-premium automotive gaming and human-machine interface (HMI) platforms. They enable higher efficiency when rendering complex curved surfaces such as car bodywork, making them an ideal fit for photorealistic ray-traced car rendering for the perfect representation of vehicles across dash-wide displays. The Photon architecture is tightly integrated into our IMG CXT and DXT GPU, which can be deployed alongside our functionally safe IMG BXS GPUs, which have been designed using processes that conform to the ISO 26262 standard for automotive safety. These can support advanced driver assistance systems (ADAS) and autonomous driving – all enabled by Imagination IP.

Introducing the IMG DXT GPU

A high-level view of the IMG DXT GPU can be seen in the diagram below:



The main components of the GPU include:

- Unified Shading Cluster (USC) – the compute heart of the GPU, a multi-threaded programmable single instruction multiple data (SIMT) processor which can simultaneously process pixel data, geometry data, compute data as well as 2D/copy housekeeping tasks. More USCs equates to higher compute performance for the GPU configuration.
- Texture Processing Unit (TPU) – handles texture addressing, sampling and filtering in highly optimised logic. More texture unit equates to higher visual complexity and higher refresh rates and display resolution support.
- Raster/Geometry block – a collection of fixed-function units enabling post- and pre-processing of data by the USC including culling, clipping, tiling, compression, decompression, iteration, etc.
- Top level – including L3 cache, AXI Bus interfaces and RISC-V firmware processor

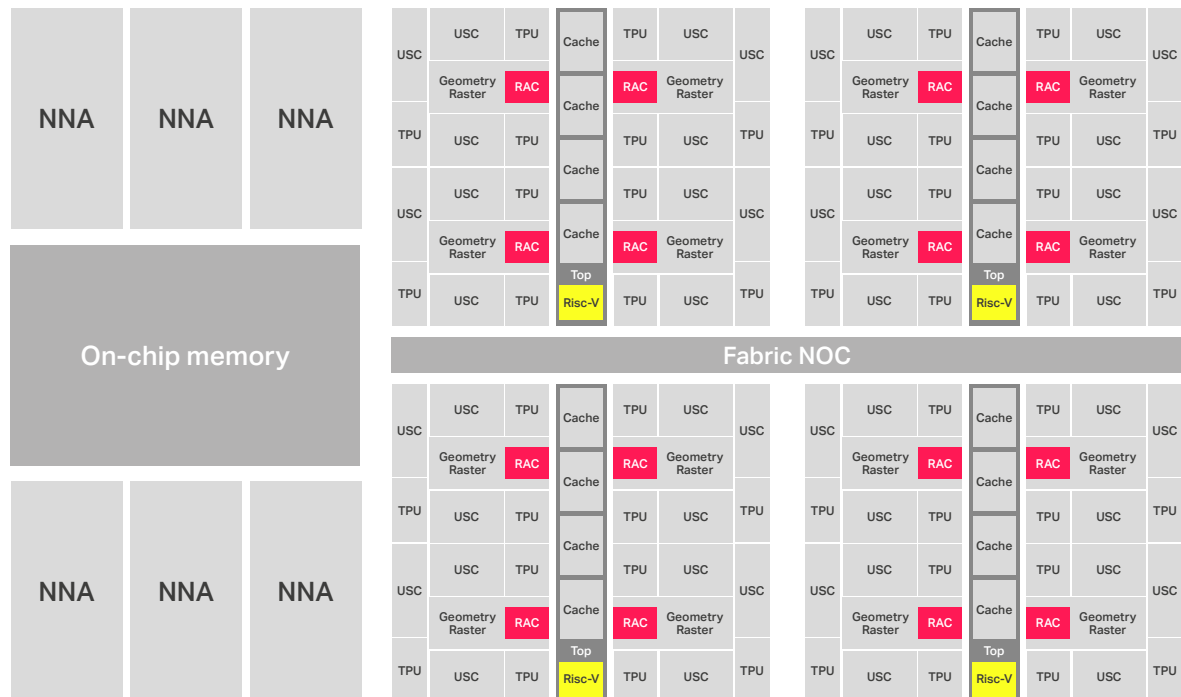
Compared to the previous C-Series GPUs the DXT contains 50% more ALU and TPU performance in a single-core unit and more flexibility in the size and location of the ray tracing block.

- Ray Acceleration Cluster (RAC) – a dedicated block for efficient handling of all ray tracing processing stages.

Introducing the IMG DXT GPU

Beyond Desktop

Similar to the previous GPU series the DXT is also available in a “beyond desktop” configuration, thanks to multi-core scaling up to four cores.



In the above “beyond desktop” configuration, the design includes optional additional IP blocks.

- **NNA:** our neural network acceleration units provide power-, performance-, and efficiency-optimised neural network processing. These units can efficiently co-work with the GPU and offer up to 100 TOPS of AI performance in a multi-core configuration with up to eight cores (six shown above).
- **On-chip memory:** on-chip shared memory, which can be used to efficiently exchange data between the GPU and NNA units. Can also be used for interaction with other IP blocks by keeping data on chip for the highest throughput, lowest latency and best power efficiency.

Introducing the IMG DXT GPU

A summary of GPU features for the above two configurations can be found in the table below:

GPU Feature	Mainstream Configuration	High Configuration	Premium Configuration
Architecture	DXT	DXT	DXT
GPU Type	TBDR	TBDR	TBDR
Ray Tracing Level	Level 4	Level 4	Level 4
Configuration	DXT-8-256 RTx	DXT-48-1536 RTx	DXT-72-2304 RTx
Scalable Processing Units (SPU)	1	2	3
Multi-Core	1x	1x	1x
Cache Size Configurable	128 - 512KB	768-2048KB	1024-2048KB
Ray Acceleration Clusters	Flexible 1x½ (RT½) or 1x1 (RT1)	Flexible 2x½ (RT1) or 2x1 (RT2)	Flexible 3x½ or 3x1 (RT3)
ALU Clusters (USC)	1	6	9
Texture Processing Units (TPU)	1	6	9
Peak 3D Texels per Clock	8	48	72
Peak Post Proc. Texels/Clocks	16	96	144
Peak FP32 Ops per Clock	256	1536	2304
Peak INT8 Ops per Clock	1024	6144	9216
Peak Blend Rate per Clock	8	48	72
Peak Tri-Ray Tests/Clock	2 per full RAC	2 per full RAC	2 per full RAC
Peak Box-Ray Tests/Clock	16 per full RAC	16 per full RC	16 per full RC
Typical Clock	Up to 1GHz	Up to 1GHz	Up to 1GHz
Peak FP32 TFLOPS	0.256	1.536	2.304
Peak AI 8b TOPS	1.0	6.144	9.216
Peak GTexels/Sec	8 to 16	48 to 96	72 to 144
Process Node	Soft IP allows targeting of 12nm, 7nm, 5nm, 4nm, 3nm, etc.		
Available	IP available for licensing now		
Customers	Multiple, across diverse target markets		

It should be noted that our GPU architecture is highly scalable and configurable to meet a wide range of market and customer requirements. The above configurations are examples only.

PowerVR Photon Ray Acceleration Cluster

Since the early days of 3D, traditional rendering has been performed using rasterisation, where the geometry of objects is built up using a mesh of triangles and then 'shaded' to create their appearance. However, with rasterisation, the way the world is lit can only be approximated.

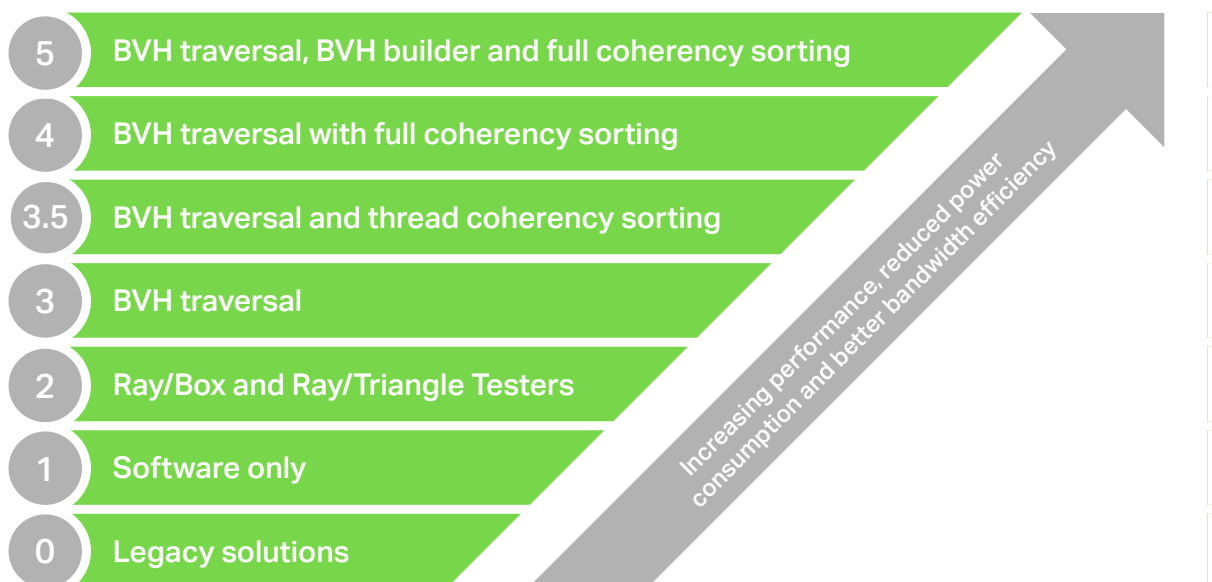
Ray tracing is different. It mimics how light works in the real world, where photons are emitted from a light source and bounced around the scene until they reach the eye of the viewer. Ray tracing sends out rays from the viewer (the screen) into the scene, onto objects and from there to the light source. As the light interacts with objects it is blocked, reflected, or refracted by the objects along the way, depending on their material properties, creating shadows and reflections, even from off-screen objects. Once the rays are fired into the scene the lighting process occurs naturally, which means developers do not have to spend time creating, "fake" lighting effects.

This elegant approach to lighting scenes helps delivers graphics with far greater realism, improving games and visual applications while simplifying the lighting process for content creators.

To better understand the fundamentals of ray tracing read our white paper, ["Shining a Light on Ray Tracing"](#).

Ray Tracing Efficiency

Ray tracing is a technology that is widely considered to be the next step in graphics technology. Since the end of 2020, it has been widely available to consumers on both desktop personal computers and consoles. To highlight the differences between different types of ray tracing acceleration Imagination created a white paper which introduced the concept of ray tracing levels to make clear that not all ray tracing solutions are created equally, and that higher-level ray tracing is more capable and feature-rich than lower levels. As the levels are incremental this white paper introduces the architectural changes and capabilities as we build up from Level 0 to Level 5. To identify your level of ray tracing [download our white paper, "What's Your Level?"](#).



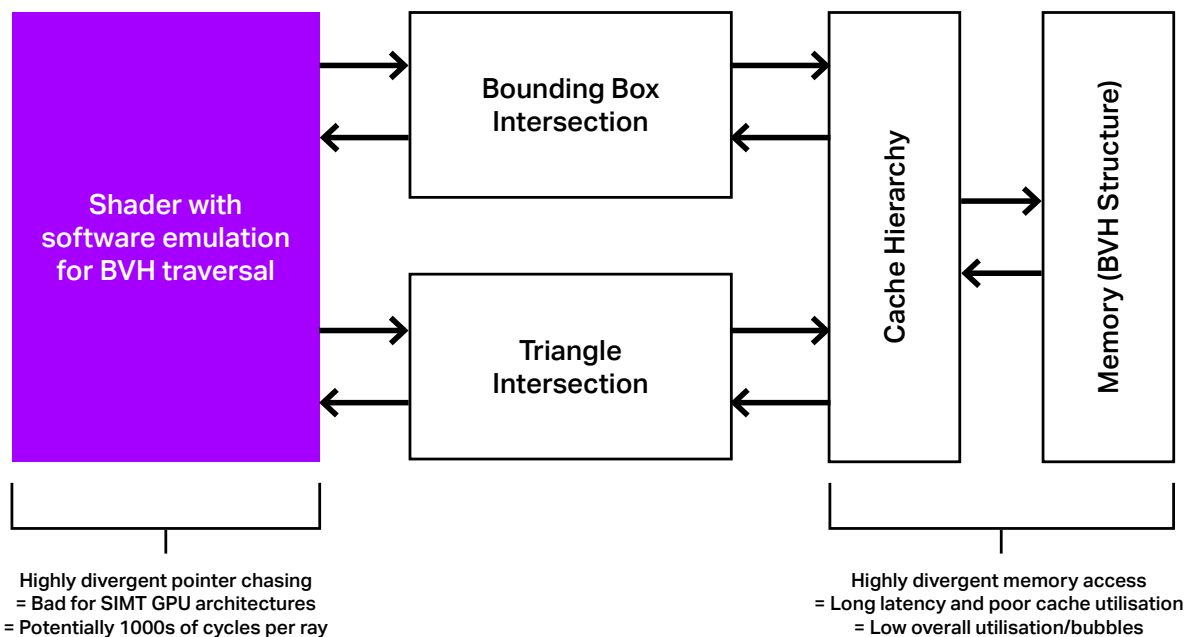
IMG Photon: delivering state-of-the-art ray tracing

Not all forms of ray tracing support are equal. The reality is that any compute-capable GPU can perform ray tracing, but what differentiates them is the efficiency with which it can be executed and the performance level which can be achieved. For this reason, Imagination introduced the concept of the Ray Tracing Levels System (RTLS) to highlight the various levels of ray tracing efficiency.

Level 1 is full software/shader-based ray tracing. This indicates how much offloading is possible from the ALU pipelines which do all the heavy lifting with compute-based ray tracing.

A Level 2 solution offloads the bulk of the ray intersection compute cost from the ALU pipelines, and this is where the concept of ray testing versus bounding boxes and triangles inside a bounding volume hierarchy (BVH) structure is introduced. This intersection-testing work requires many instructions, and these can be handled much more efficiently in a fixed-function hardware block dedicated to the task.

Level 2 Ray Tracing – Adding Box/Triangle Testers



The problem with this approach is that all the other ray tracing work remains on the shader cores, and, unfortunately, this is highly divergent code with many branches, since based on each bounding box intersection you decide which lower-level boxes, and ultimately, triangles, you need to test. Branches are inefficient to process on parallel compute engines, such as GPUs, hence this process is not a good fit for execution on the shader engines.

Hence for every ray cast, multiple levels of bounding box data have to be fetched and tested – and then, based on hits or misses, more data has to be identified and fetched from memory. Effectively then, the whole traversal of the BVH structure is done in shader code that is branch

PowerVR Photon Ray Acceleration Cluster

heavy and divergent, which is not a good fit for the same instruction multiple threads architecture (SIMT) of the GPU.

To compound matters, the complexity of the BVH structure grows with more complex scenes which means that many levels of boxes and triangles will have to be tested to find the hit (or miss) for each ray. This count is not the same for each ray (e.g., some rays may miss immediately, while others may take many levels to do so), which again is a poor computational fit for the parallel processing architecture of the GPU, leading to a lot of lost performance.

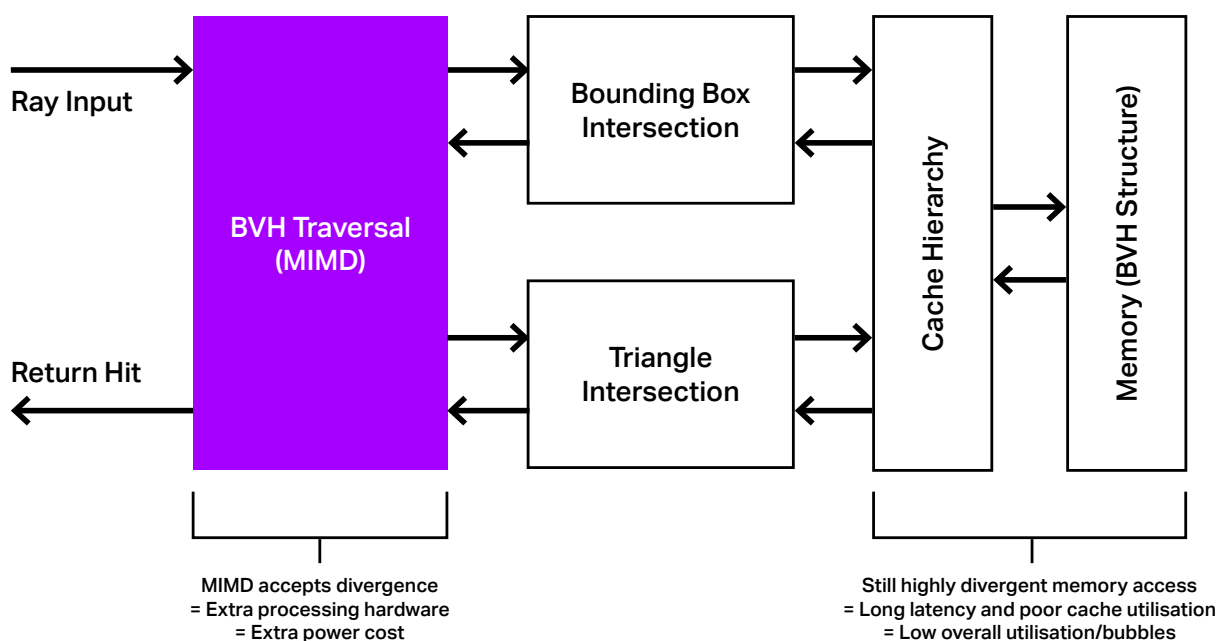
Worse still, as each ray can go in a different direction and take a different path through the BVH structure, the data access is also divergent and thus non-coherent. This means that cache hits will be very low, and the GPU will suffer very long latencies before box or triangle/ray intersection tests can be executed, which again will lead to pipeline bubbles and low processing efficiency.

From all of this it can be understood that while some of the arithmetic cost is offloaded from the shader cores, they are still inefficient for ray tracing, as too much work remains in software on them and is of a nature that does not fit well with the optimisation strategy of the GPU.

Hence this approach is a Level 2 RTLS solution – better than software, but unlikely to be efficient, and, when considering mobile solutions, unlikely to be usable in real-world power and bandwidth budgets.

Level 3 RTLS – Full Hardware BVH Traversal

In a typical Level 3 RTLS solution (illustrated below) the BVH traversal moves from the GPU ALU pipelines into a dedicated processing unit. As the nature of the BVH walking remains divergent, the most common approach to handling this is to adopt a multiple instruction multiple data (MIMD). architecture.



PowerVR Photon Ray Acceleration Cluster

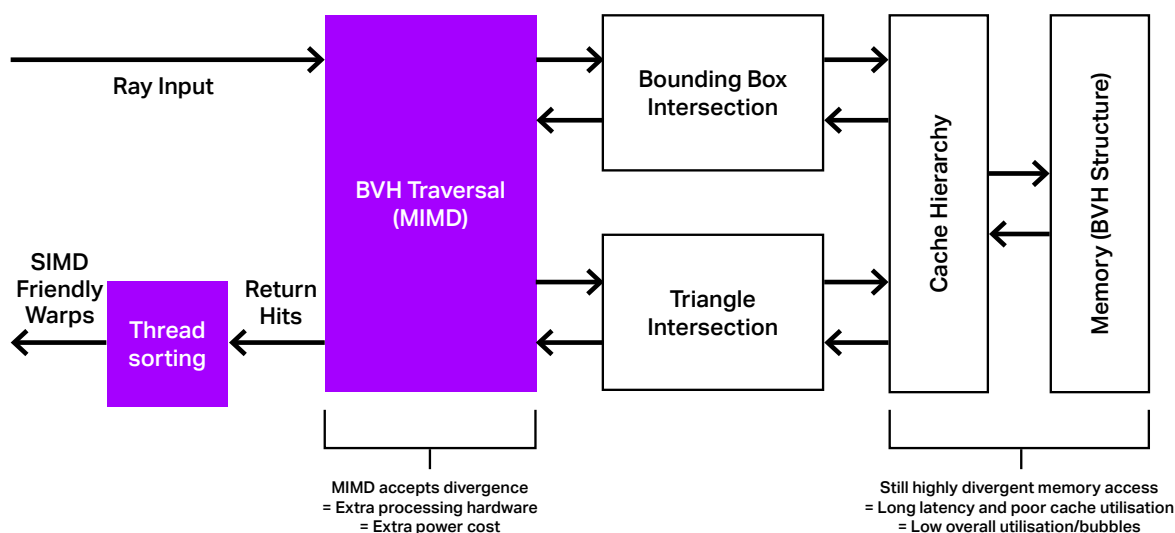
This is a design that accepts divergence (where each ray is different from all the other rays and hence requires its own logic) as part of the processing design. This costs silicon area but also power and is very different from the typical way GPUs achieve their efficiency and high performance, which is lots of units all doing the same thing (parallelism).

While a MIMD design is parallel, it is more like a many-core CPU design where each core does something different. This executes efficiently – e.g., every unit does work, but in area and power cost it is linear, as you replicate all logic for every unit. The benefit, versus the usage of the ALU cores, is that each unit will do useful work each clock, wherein the SIMT GPU design many lanes will go idle due to the divergence.

This is a higher level of efficiency, as the BVH walking on the MIMD unit will be more effective than trying to run it on the SIMT structure of the GPU ALU pipelines. Unfortunately, while the MIMD unit enables divergent processing to gain efficiency (be it area and power costs) the divergence of the data remains a fact. As the name says, it's "multiple data", which, just as in the Level 2 RTLS solution, puts a lot of strain on the cache hierarchy and the memory subsystem. This is because this data access remains divergent, resulting in poor cache hits and, ultimately, high bandwidth, power, and efficiency costs. It also has a risk of pipeline "bubbles" (a delay in execution of an instruction), which is where the box/triangle testers, BVH traversal units and shader pipelines will stall, as the job will be limited by the memory speed.

Again, using this type of architecture in smartphones is unlikely to be effective, as unlike desktops, their GPUs have minimal bandwidth budgets, and, worse, as there is no GPU-specific memory, they share that bandwidth budget with the CPU and other processing units. They instead rely on an architecture with unified memory, which means shared bandwidth and even longer latencies. Some desktop GPUs have 128MB of on-chip cache to help, claiming that most of the BVH structure will fit into this memory but large caches have large silicon area costs and cost power to operate. Again, this type of dedicated buffering, a brute-force solution, is simply not viable in a smartphone product, where a premium GPU typically has a 2MB cache. As such, a superior, higher-efficiency solution to the problem of divergence is essential.

The new Level 3.5 Solutions



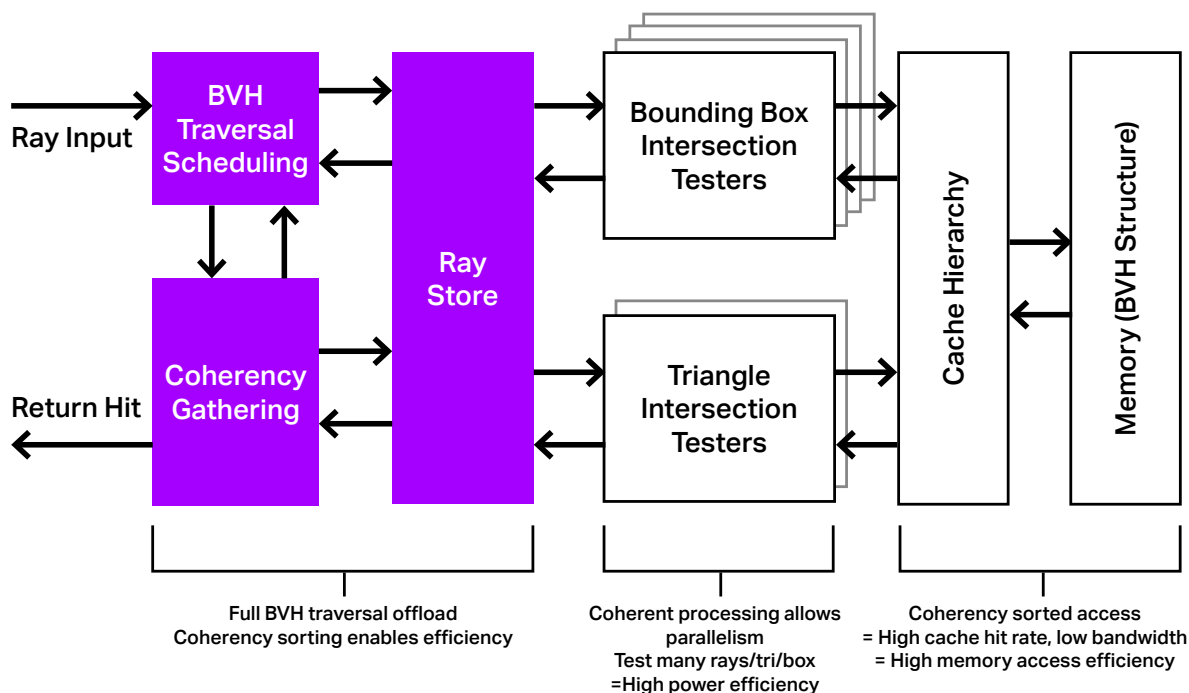
PowerVR Photon Ray Acceleration Cluster

In 2022, new ray tracing solutions entered the market, which are a step up from the Level 3 solution described above, but still behind the Level 4 solution discussed in the next section – hence the 3.5 classification.

Fundamentally, the actual ray processing of the Level 3.5 solution is identical to the Level 3 solution, and this includes all the disadvantages. This means that the BVH traversal continues to be non-coherent and the BVH data fetches and processing continues to suffer from divergence, both in memory access, which results in cache misses and power inefficiency, but also in processing.

What the Level 3.5 solution does differently is the handing back of the hits to the shader execution unit. Where a Level 3 solution returns hits as they are, which means divergent and a mismatch with the efficiency requirements of the SIMT execution nature of the GPU ALUs, the Level 3.5 solution inserts a thread sorting block which regroups the threads into warps with more optimal SIMT execution characteristics and hence delivers much improved ALU utilisation within the GPU itself. There is no doubt this will deliver a marked efficiency improvement, but it leaves the ray coherency problem itself untouched.

Level 4 Ray Tracing Levels System



PowerVR has always been synonymous with efficiency using the ultra-efficient tile-based deferred rendering approach and avoiding the brute force immediate mode rendering strategy. In developing our ray tracing solution, first demonstrated in 2016, we have taken the same approach. The PowerVR Photon architecture, further evolved for 2023, is the latest incarnation of this technology, designed to enable ray tracing in smartphone power and bandwidth budgets,

PowerVR Photon Ray Acceleration Cluster

while also allowing this efficiency to be scaled up into markets beyond mobile.

The problem at the core of ray tracing is a lack of coherency; rays can, and will, go in random directions. The problem is that this clashes with the parallelism designed into traditional GPUs. The best solution to tackle this issue is by focusing on the workload, and for this, we introduce a coherency gathering unit.

With this unit, the BVH walking is still fully offloaded, but it now becomes a scheduling problem. We have many rays which we can store, and the coherency unit then looks to group rays into packets or bundles which are similar – as in, rays that take similar paths through the BVH acceleration structure. These are said to be “coherent”. While they may be non-coherent from one ray to the next, averaged across many rays there are always similarities and correlations which we can exploit, and this is exactly what the Photon architecture does.

In it, rays are grouped into processing packets that will achieve high efficiency, not only in processing but also in memory access. This sorting gives us another benefit: rather than a MIMD architecture we return instead to the high-efficiency processing approach common inside the GPU: many units which all do the same thing.

As a result, we can exploit parallelism, as we do not just check one ray against one box, we can check many rays against the same box. This brings significant efficiency gains and reduces stress on the cache and memory subsystems. The same is true for triangle intersections: we can check a ray against multiple triangles concurrently.

There are, therefore, four fundamental benefits of the Photon architecture.

- Full BVH traversal and box/tri-testing offload from the ALU pipelines.
- Coherency gathering means that ray processing becomes parallel.
- Coherency means that data reuse is very high and the stress on cache and memory subsystems is significantly reduced.
- With many rays in flight, the ALU shading work and ray tracing can be decoupled, and with many elements in flight, latency absorption becomes effective.

The above concepts have been in proven hardware designs used for internal testing and analysis in Imagination since 2016. The results are now on show inside our Photon architecture featured within our latest generation of GPUs, bringing hyper-efficient ray tracing to our mobile customers as well as other markets.

PowerVR Photon Ray Acceleration Cluster

MIMD versus SIMT

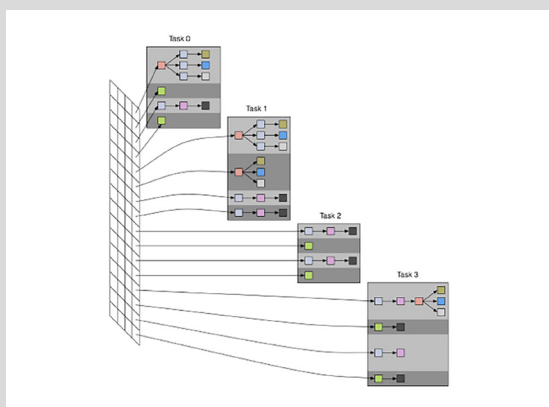
Multiple instructions, multiple data, (MIMD), is an idiosyncratic type of parallel processing architecture designed to handle processing which, in effect, is not parallel in nature. As the name suggests, it involves different instructions (so different processing) and multiple data (so different information to work on). The problem with this is that to make MIMD fast you end up with many fully-featured individual processing units, all of which are designed to work individually with very little shared, as, per the MIMD nature, there is no benefit from sharing anything since everything is different by definition.

A massively multi-core CPU design is effectively MIMD and while this can process parallel compute jobs, such as neural networks for AI, we also know this is not area, power or bandwidth efficient compared to truly parallel designs.

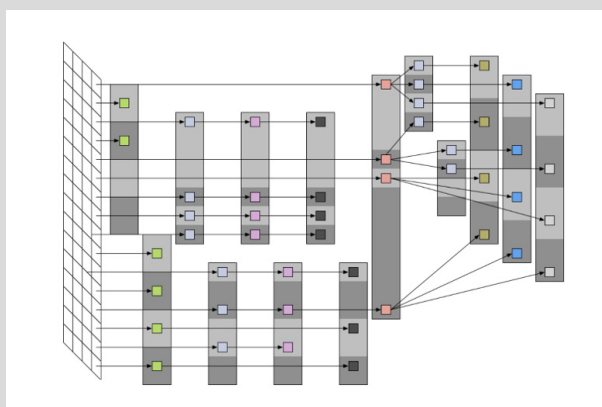
GPUs by design are typically described as having an architecture that uses the same instruction across multiple threads, (known as same instruction, multiple threads or SIMT) in nature, which means they execute the same work but on different data elements. These are typically vertices for geometry and pixels/fragments for shading, and now, for ray tracing. SIMT is highly efficient because we can deploy many units which all do the same work and hence can share a lot of logic since to execute the same job there is a lot in common. SIMT, therefore, provides significant logic and processing efficiency gains.

However, by accepting MIMD in your design, you essentially accept defeat and acknowledge that what you process is not parallel and that, as such, all the benefits of this parallelism are lost. The result is highly divergent needs, which means more silicon area and increased power consumption. In a MIMD structure, processing can also go out of sync, since one element may finish processing virtually immediately, (e.g. if a ray misses the first bounding box in the BVH) or it could take many 10s or even 100s of cycles of processing (e.g. say a ray which goes deep down the BVH structure to intersect a triangle).

The problem with these different depths is that for the ray query to return to the SIMT ALU inside the GPU all rays need to finish and return to their warp. MIMD does not resolve this unless the whole GPU becomes MIMD, but then it loses all of its benefits and effectively becomes a massively parallel CPU design. Hence MIMD is parallel, but, in terms of the whole solution, not really, and therefore, is not an efficient nor smart solution to the processing problem of ray tracing.



Competitor MIMD Execution, four MIMD processors. Note how each processor executes with different stages and data (colours).



PowerVR SIMD style execution following coherency sorting. Note how each cycle the units execute the same instruction and use some of the same data.

PowerVR Photon Ray Acceleration Cluster

Ray Tracing Efficiency Summary

	Level 2	Level 3	Level 3.5	Level 4
Example implementations	AMD GPUs/ Console GPUs	NVIDIA <RTX40-Series	Nvidia RTX40- Series Intel ARC	PowerVR GPUs
ALU Offloading	Partial	Full	Full	Full
Hardware Box Testers	Y	Y	Y	Y
Hardware Triangle Testers	Y	Y	Y	Y
Hardware BVH Processing	N	Y	Y	Y
Ray Coherency Sort	N	N	N	Y
Thread Coherency Sort	N	N	Y	Y
Cache Hit Rate	Low	Low/Medium	Low/Medium	High
Memory Latency Tolerance	Low	Low	Low	High
Processing Efficiency ALU	Low (SIMT utilisation)	Low (SIMT utilisation)	High (Thread Sorting)	Very High (Coherency Gathering)
Mobile Power Budget	No	No	No	Yes

The Photon Architecture RAC In-Depth

Introducing the RAC – Ray Acceleration Cluster

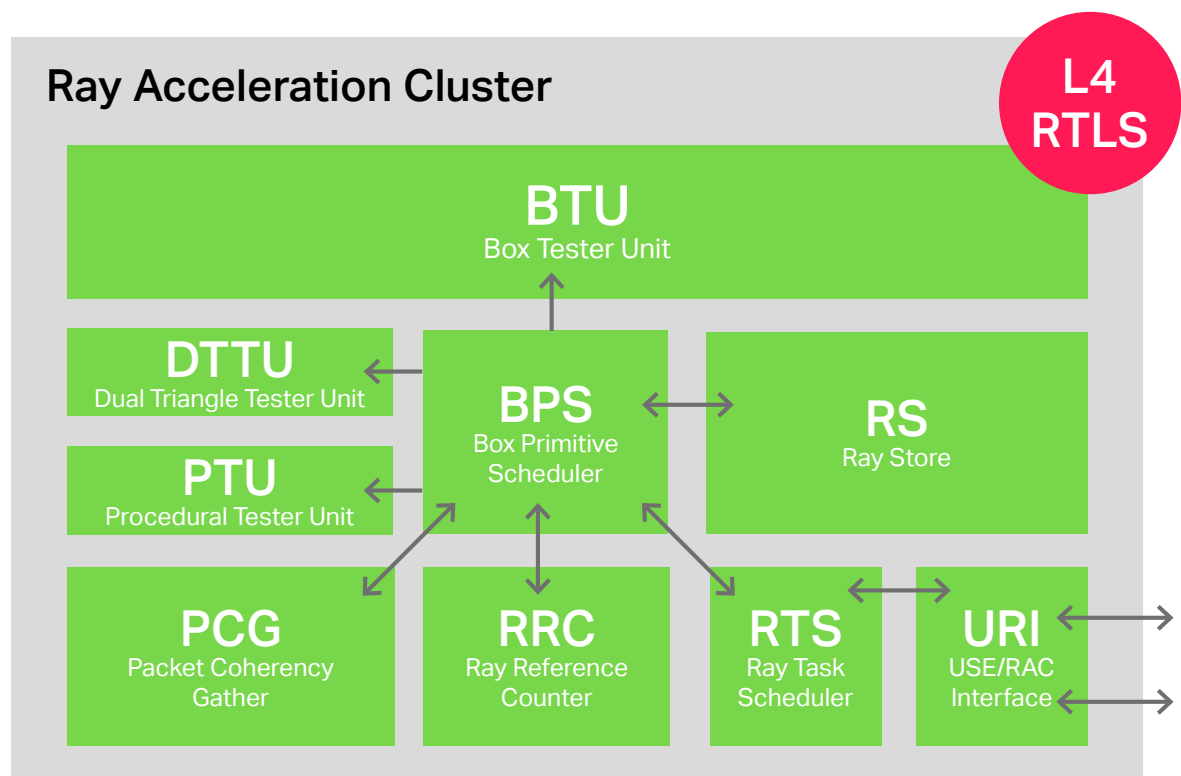
The PowerVR Photon architecture adds a new block into the PowerVR GPU called the ray acceleration cluster or RAC, which is responsible for all ray tracing activity on the PowerVR GPU. This encompasses the entire process; from emitting a ray (from a shader/kernel) to returning the hit (or miss) results back to the ALU for processing.

As rays are generated, and results processed, by graphics shader or compute kernel programs the RAC is very closely coupled to the GPU's ALU engines. While the units are closely linked to exchange ray and hit/miss information they are technically fully "decoupled", meaning that both units operate concurrently to achieve the highest possible efficiency and utilisation.

The RAC effectively handles the full BVH walking, including the very computationally intensive box/ray and triangle ray intersections, as well as efficiency optimisations such as the coherency sorting. The RAC is fully compatible with all modes and functionality exposed by the current ray tracing APIs, including Vulkan® extensions and DirectX ray tracing.

The RAC is a scalable unit supporting multiple performance points (e.g., 1x, 0.5x, 0.25x of a RAC) as well as multi-core scalability (2x and beyond), where multiple RACs can be placed next to the ALU unit(s). Within the current PowerVR GPU design a RAC is shared by two 128-wide ALU units, enabling increased utilisation of the RAC, ALU and texture processing unit (TPU).

The combination of RAC, three ALUs and three TPU units with scheduling logic and other fixed-function support is called the Scalable Processing Unit (SPU). This forms the basic unit that builds our range of CXT GPUs from one up to four SPU units per GPU core, which can then scale even further thanks to our decentralised multi-core system.



Building Blocks

Ray Testers

Each RAC contains several ray testing units – these are fixed-function blocks that are responsible for checking if there is an intersection between the ray and the respective primitive, be it a bounding box that forms the bulk of the bounding volume hierarchy (BVH), triangles or procedural primitives.

As the acceleration structure consists mostly of a hierarchy of bounding boxes, the number of box testers is logically the largest, and this testing extensively uses parallelism for the highest efficiency; e.g. testing rays/boxes in bulk. The triangle testers are next, and, as triangles are by far the most popular primitive for building 3D geometry, this type of testing is highly optimised in hardware and also parallel in nature. As shown in the block diagram we deploy a “Dual Triangle Testing Unit”, which means we test each ray against two triangles, again, taking advantage of parallelism for optimal density and power and performance efficiency.

Last is a unit for handling procedural tests. This is supported, but as indicated by the APIs, is slower but much more flexible as here a special intersection shader program has to be executed to handle the testing. The exact count of units and ratios depends on the target performance level of the RAC (e.g., more units for higher-performance RAC designs).

Ray Store Unit

The next unit is the Ray Store Unit and as we want to keep both the ALU and the RAC fully loaded we need storage to allow these units to operate concurrency. This storage is also essential to allow for coherency sorting (see next) and, as such, the Ray Store unit holds ray data structures during processing, keeps data on-chip and provides high-bandwidth read/write access to all units in the core.

Packet Coherency Gathering Unit

The Packet Coherency Gathering unit is a unique and patent-protected block that makes the Photon architecture a Level 4 RTLS solution. It is responsible for analysing all active rays and creating packets (groups) of coherent rays (rays with similar trajectories) to test against the scene together. By testing and processing coherent rays, we enable parallelism and achieve higher processing and bandwidth efficiency, as thanks to data re-use we can achieve high hit rates on our cache structure.

Ray Task Scheduler

The Ray Task Scheduler is responsible for managing the interaction between the ALU in the GPU core handling resource allocation and coordination and communication between all processing stages within the RAC as well as coordination with the ALU/USC inside the GPU.

Box Primitive Scheduler

The Box Primitive Scheduler is the hardware that manages the process of walking the BVH structure, triggering box, triangle and procedural intersection tests and the follow-on work as required by the results of each operation.

Ray Reference Counter

The ray reference counter is an essential part of the scheduling activities and its role is to keep track of the work per ray that is still pending and check that all ray testing is done so the result can be sent back to the ALU. As rays need processing this is an incrementing counter, and, as the jobs generated as part of the BVH testing complete the counter is decremented until you end at zero when all tests are complete. Tracking what is in flight and what is done is essential due to the coherency shuffling, which groups rays into different packets based on coherency for processing.

USC/RAC Interface

The USC/RAC interface is the data movement engine between the ALU units - also called unified scalable clusters (USCs) - in the PowerVR GPUs and the RAC. This unit receives ray requests and sends ray results across this interface. As in all processing, effectively and efficiently handling data movement is key to avoiding bottlenecks.

Power Efficiency Focus

Power efficiency is king today, not just in mobile, where it drives battery life but increasingly also with regards to thermals, which again can impact smartphones where you do not want to burn your hand while running a AAA-game or suffering thermal throttling where suddenly the user experience becomes poor due to low clock frequencies and hence lower framerate. Thermals are also critical in server environments where the density of the racks is very much thermally constrained. As such, power efficiency has always been at the forefront of all architectural choices made by Imagination in developing our PowerVR architectures.

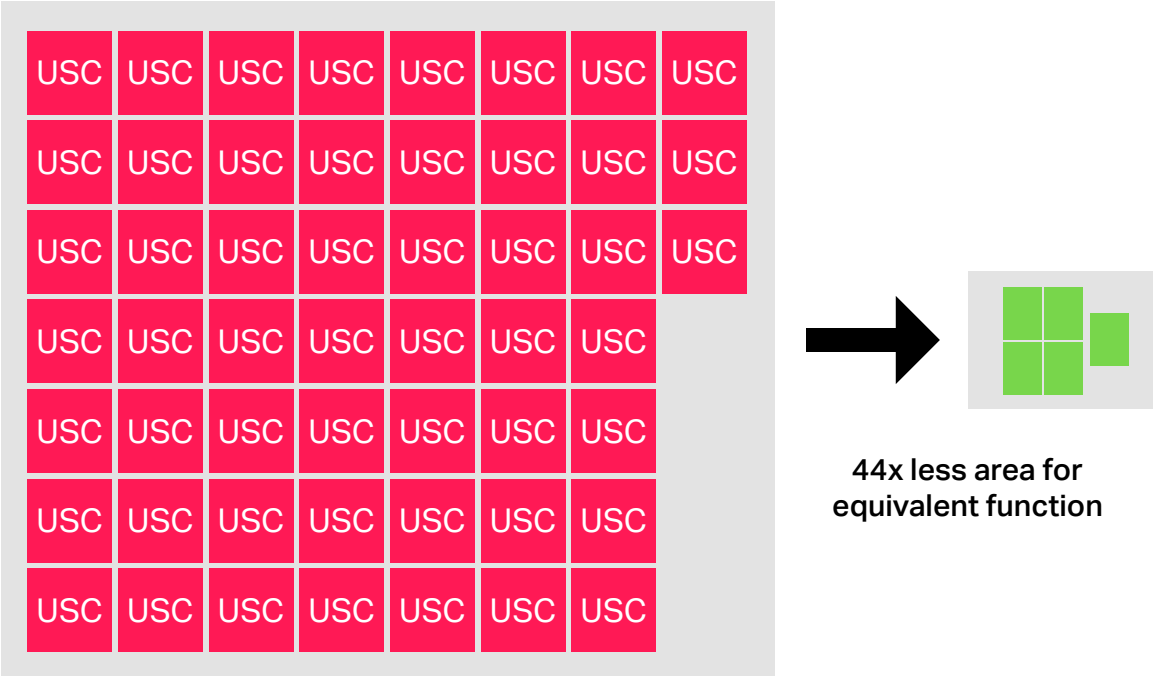
The Ray Tracing Level System is also very much about power and performance efficiency and the key thing for ray tracing is understanding just how much more power-efficient fixed-function/ specialised hardware is versus programmable hardware. Fundamentally, programmability offers ultimate flexibility, and this can be extremely powerful and valuable, but that flexibility, like anything in life, comes at a cost and that cost is power efficiency and density. We learned this in the early stages of our ray tracing history, where we worked out how many USCs, we would need to handle the ray-box intersections at the same speed as a relatively modest dedicated fixed-function unit. A visual representation of this can be seen here:

Essentially, just doing a ray-box intersection test using programmable GPU logic, essentially fused multiple adds (FMA), requires 44x more silicon area than what can be achieved in a fixed-function block which offers the same capability. This is what a Level 2 RTLS solution focuses on – moving these computationally expensive ray-box and ray-triangle intersections into dedicated specialised more efficient hardware.

Now, as already explained, with a Level 3 RTLS solution you shift even more of the ray tracing workload from programmable GPU logic. Indeed, we move near all ray tracing work from the USCs/shader units and move it into more effective specialised hardware. With Level 4, we further boost this efficiency by enabling effective parallelism with SIMD/SIMT style approaches rather than expensive MIMD, and we also solve processing efficiency as well as memory access

Building Blocks

efficiency by grouping rays that follow similar paths through the acceleration structure.



The following table summarises the efficiency Levels and how they impact execution efficiency and the resulting power, performance and bandwidth efficiency:

GPU Block Ray Tracing Task	Level 1 RTLS	Level 2 RTLS	Level 3 RTLS	Level 4 RTLS (PowerVR)
ALU Loading	Full	High	Low	Low
ALU Efficiency	Low	Low	Medium	High
Box/Tri Testers	N/A	Medium	High	Full
BVH Walking	N/A	N/A	Yes	Yes
Coherency	No	No	No	Yes
Cache Hits	Low	Low	Low/Mid	High
Bandwidth Usage	High	High	Mid	Low
Power Efficiency	Very Low	Low	Medium	High

Confusing Numbers

New hardware capabilities always trigger new metrics that try and indicate the performance level of the feature, and ray tracing is no different. The first ray tracing-enabled GPUs launched into the desktop market promoted a gigarays per second (GRays/s) number; a logical concept, because as we are tracing rays it would be good to have a sense of the hardware's capabilities at doing this. Unfortunately, this is a problematic metric since a ray could miss the top-level scene bounding box and immediately return, or a ray could go down the rabbit hole of hundreds of box and triangle tests before it hits anything – so the number of rays you fire doesn't equate to real-world performance. Essentially, if not all rays are equal in complexity, then how can we talk sensibly about GRays/s?

Later product launches saw a more classic approach to quoting performance which was to quote the box tests per second and triangle tests per second. While this is a more useful and factual metric, it still raises a lot of questions. What is the correct ratio between those rates? How efficient are those units and can they be used all the time? Or are they limited by maybe cache hits or other data flow issues?

Finally, we have even seen ray tracing operations per second (RTOPs) claims, which is a made-up metric that tries to translate the fixed-function capabilities of ray tracing hardware back to equivalent shader instruction costs and claims a random cost of 1000 ALU operations per ray. It makes for a nice big marketing number, but, as with GRays/s, is not meaningful, as the number is arbitrary, and the cost could easily be higher or lower.

The Reality of Ray Tracing Benchmarks

Unfortunately, most GPU metrics have limited meaning, as most are poorly defined and lack context or specific benchmark conditions. The new ray tracing metrics are no different in this respect. While these metrics have some value – they usually reference the peak capability of a hardware processing unit – in real-world usage, they will vary wildly in efficiency and utilisation, as the peak number metric does not indicate its efficiency.

The ray rate in GRays/s is a good example of this. It's just too vague and generates a "my number is bigger than your number" race. In reality, GRays/s will be influenced by the scene: larger scenes will have deeper hierarchies with more boxes and triangles to test in the hierarchy to find the hit/miss, resulting in a lower giga-ray rate, depending on how complex your test scene is.

This is why original gigarays per second number claims were in fact, "gigamiss" per second rates, as the claimed number was for a scene where all rays would immediately miss the whole scene. This is a bit like quoting game performance while staring at the floor or sky with nothing complex in view. The box and triangle test rates are slightly more valuable but ignore the utilisation efficiency of these units, especially in a Level 2 RTLS solution, where it's likely that the processing will be bandwidth and/or shader bound, and not test bound – so again those numbers become a theoretical peak not a real usable number.

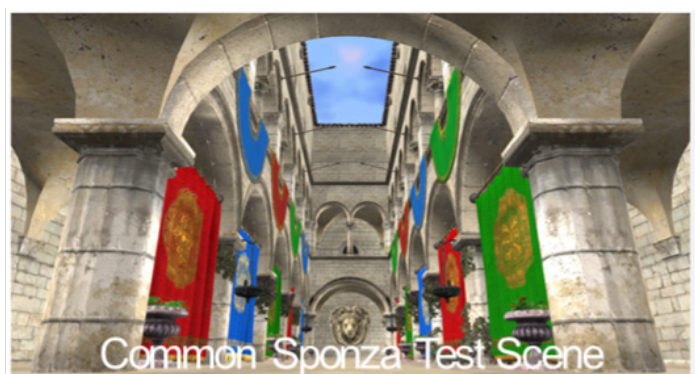
Unfortunately, that leaves us with benchmarks where we lock all these conditions: not only the scene complexity but the view direction and the size of each ray's payload, as the mix of ALU and ray-bound cycles will all impact the number that will come out. A good example of this is Imagination's internal work on acceleration structures, where a fast real-time builder versus a slower, but superior builder can impact the achievable performance by more than 30%, as the

Confusing Numbers

more efficient hierarchy does a better job at culling work away. Of course, benchmarks can still be misleading as the scene may be a better fit to the ray versus box ratio one vendor may have selected. The benefit of selling IP is that a lot of those ratios can be tuned quite late in the game to match market and even customer-specific expectations.

Real-World Efficiency Analysis

With multiple ray tracing solutions in silicon, we can dig into the real-world efficiency of the various ray tracing level implementations. Our internal teams have dug into a couple of workloads looking to understand the difference between Level 2, 3 and 4 solutions. The test scene of choice is the Sponza scene: a widely used reference scene used for testing many graphics and especially ray tracing algorithms.

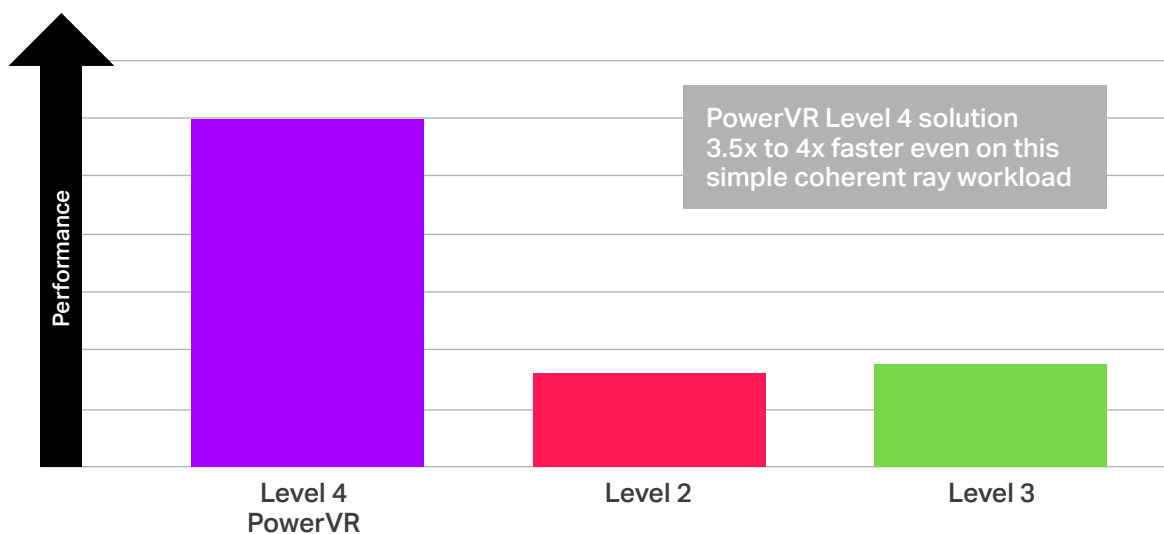


The first test case is looking at very simple hard shadows.

- For each pixel send a ray to the light source. If you hit an object, the pixel is in shadow – if you hit nothing (the light source) you are lit.

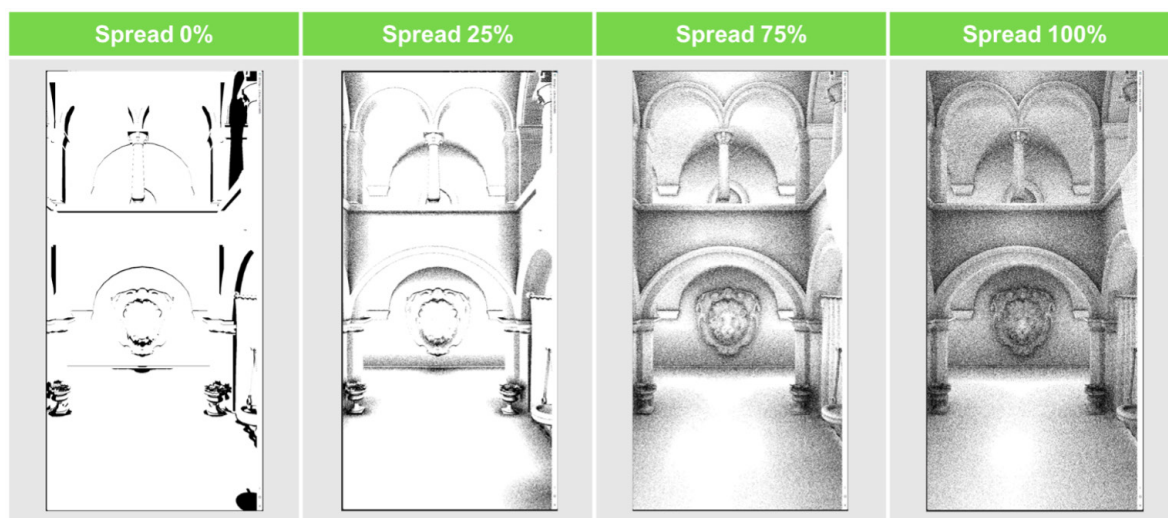
The characteristic of this workload is minimal shader workload and high coherency of rays, as naturally for each pixel, the ray is travelling towards the light source and hence there should be natural high re-use of BVH data and high cache and processing efficiency. As the shader is also very similar no matter what object is hit there should also be little to no impact of divergence. So, for this scene, overall, the ray tracing work is very simple and friendly to any type of hardware implementation and the sustained ray rates can be supported by the overall design.

Confusing Numbers



Measurements on real platforms with the latest drivers show that there is still a big gap in efficiency, whereas our Level 4 RTLS solution can sustain its quoted performance rates (approaching peak throughput for this type of simple workload). Crucially, we do not see this for the Level 2, nor Level 3 solutions, which trail by a factor of 3x to 4x.

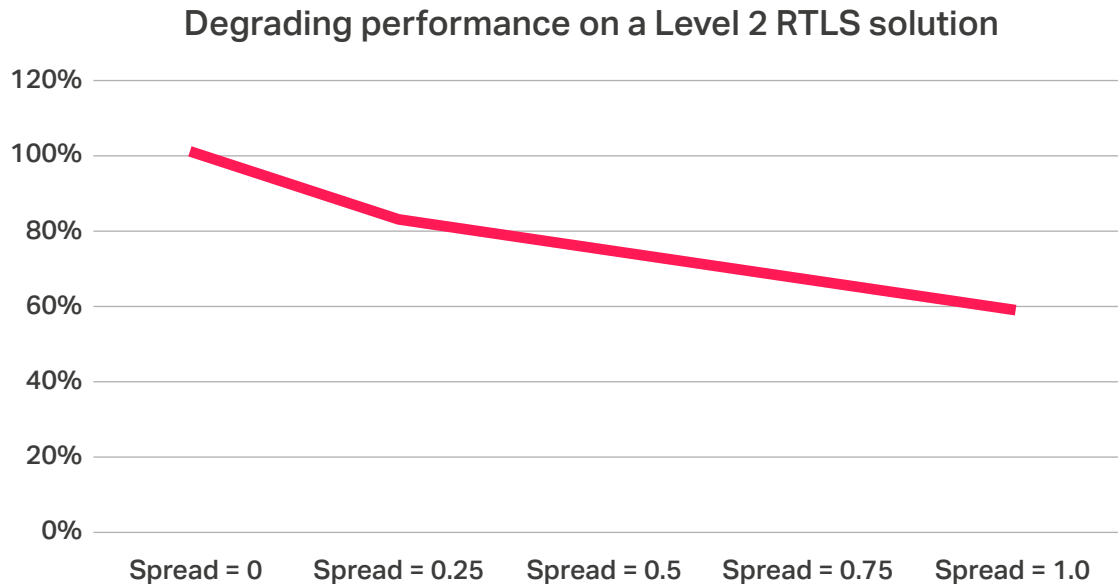
The second workload we have investigated is looking more at the impact of ray divergence. For this, we use the same Sponza scene, but we look at an ambient occlusion algorithm instead of a simple hard shadow algorithm. We still only use one ray per pixel but to capture ambient light information we ideally need to send rays in many different directions to capture the scene's light characteristics: the more distributed those rays the better the approximation, and the more coherent the worse our approximation. Hence, we can use this to test how well hardware deals with an increasing amount of ray divergence.



These images show how important the handling of divergence is for visual quality. Divergence hurts the quality of the light information collection, and while the widespread ray case is noisy it does very clearly capture the real light characteristics of the scene. With Imagination's custom denoising algorithm, this can be turned into a smooth representation of the light characteristics of the Sponza scene.

Confusing Numbers

Analysis of this scene on a Level 2 solution shows the following trend:



Again, this is a very simple scene, with minimal shader loading and minimal bandwidth usage. There are no textures so nearly all the GPU resources and caches can be assigned to speed up the ray processing. Despite this, we see a drop from 100% peak starting efficiency (fully coherent rays) to 60% of peak, with full ray divergence spread. This is as expected as the hardware includes no smarts to handle the divergence.

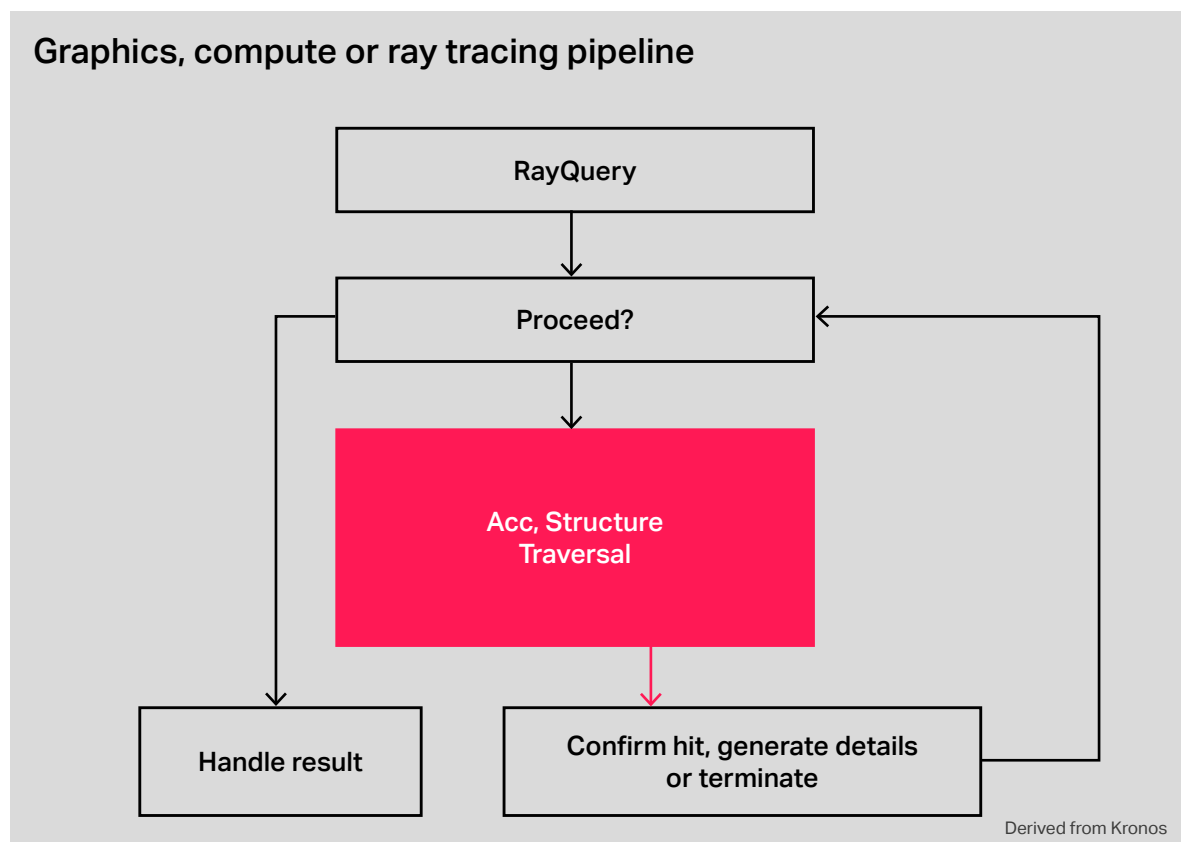
A Tale of Two Ray Tracing APIs

There are two distinct types of ray tracing which are exposed by both the Khronos Vulkan API as well as the Microsoft DirectX 12 ray tracing API. Both approaches are summarised in this section and both modes are fully supported by the PowerVR Photon architecture to ensure the highest possible content compatibility and efficiency.

VK Ray Query

Also known as inline ray tracing under Microsoft DirectX Raytracing (DXR), ray queries are quite easy to understand. Essentially, any shader or kernel (compute) can issue a RayQuery, which kicks off the ray tracing process as described in previous sections. In this system, the resulting hit/miss information returns to the same shader/kernel that has to deal with it. Therefore, the ray tracing is very simple and, as per the DXR name, is effectively an inline process.

A simple example of this would be shadow rays. Here, the scene is rendered as normal, but now in the fragment/pixel shader a ray is emitted towards the light source and when that source is hit, we know the current pixel is lit and we can execute the correct code in the shader. If we hit any other object in the scene, we know it is in shadow, and again, you can execute the correct code in the shader. Reflections would be a lot harder in this scheme since as the reflected object is hit, we have to trigger a lot of complexity to figure out how to render the correct colour for that reflected object, and this all must be handled in the original casting shader.

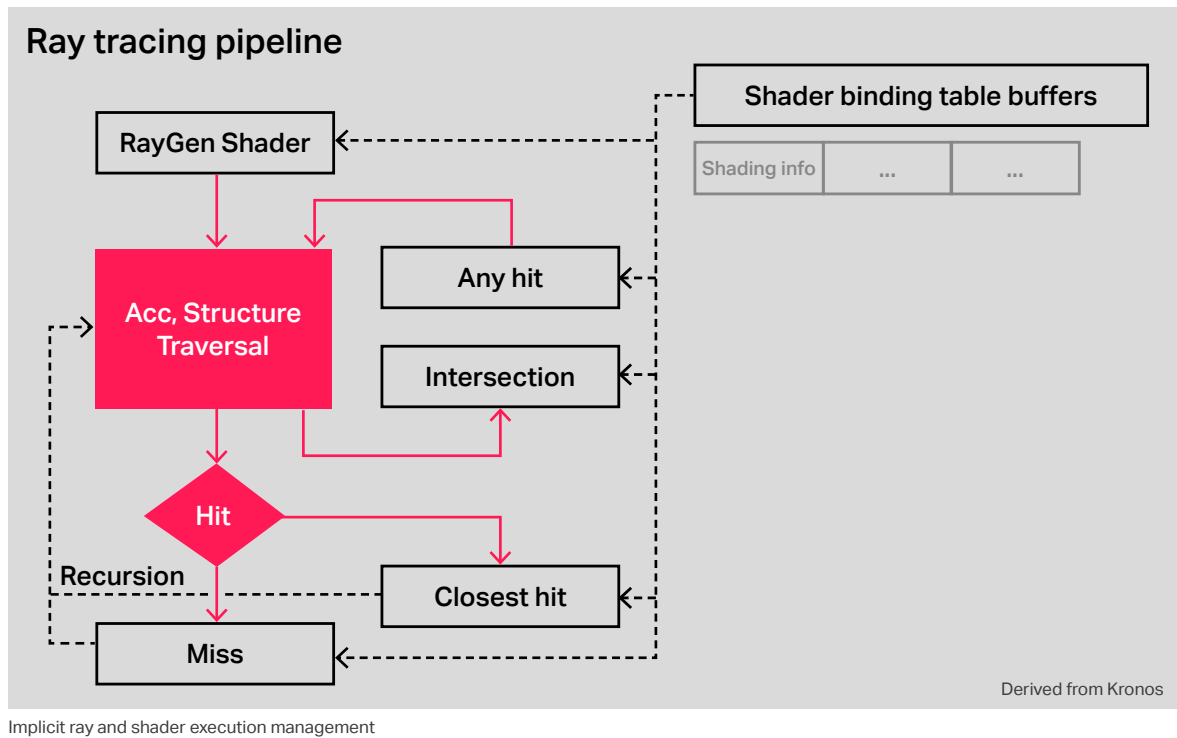


Explicit ray tracing management within single shader

A Tale of Two Ray Tracing APIs

Vulkan Khronos Ray Tracing Pipelines

Vulkan ray tracing pipelines are much more complex, and are illustrated as follows:



The ray-tracing pipeline starts from a specific RayGen shader, which is typically a camera/screen from which you cast primary rays into the scene, to start the process of generating the ray-traced image. The key difference with ray query is that hits and misses trigger recursion, so based on hit and miss events, (as shown in the diagram above), the correct shaders will be triggered and executed. This means that complex and multiple bounces (e.g., primary ray to secondary and following rays) are more easily handled and complex full ray traced concepts are easier to implement.

It should be noted that the impact of the ray pipeline on the ray tracing hardware itself is limited; e.g., both approaches trace a ray through the acceleration structure and return miss/hit information.

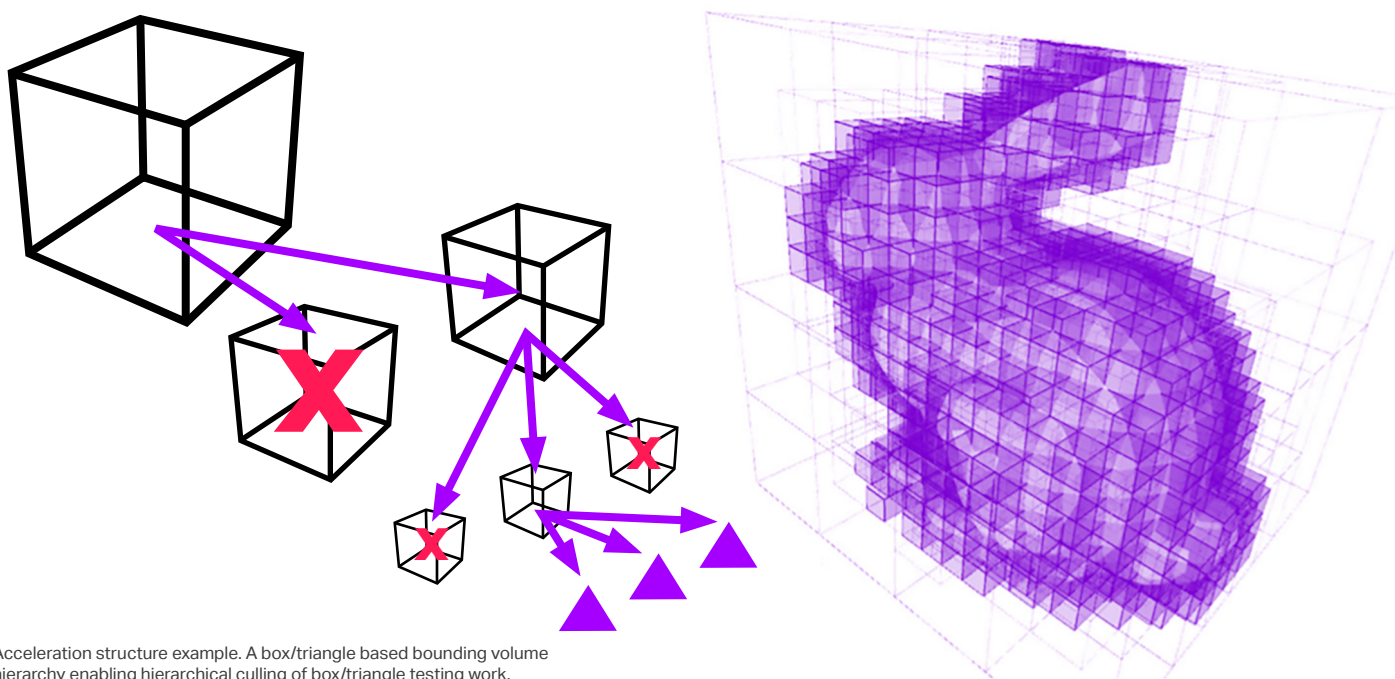
The key change is how that result is handled; explicit/manual in the shader for ray queries, and more dynamic implicit in the ray pipeline case. The main impact on handling the ray pipeline functionality is on the compute/GPU side, where recursion support is needed and the ability for the GPU to schedule its own shaders/kernels based on hit/miss information. This recursion support has interesting performance complexities as a ray could bounce through a lot of different objects, each launching more shaders/kernels and requiring more resources inside the GPU. As such, one of the performance pitfalls to be aware of is that at some point fast on-chip storage will run out and recursion will trigger an overflow to system memory, which could impact performance significantly (a cliff event where performance suddenly changes).

A Tale of Two Ray Tracing APIs

As a result, for most initial rendering algorithms, the usage of ray queries will be recommended as this is simpler to add to existing game engines and is also likely to offer more predictable performance across implementations.

Acceleration Structure Concept

Throughout this whitepaper we have made references to acceleration structure and bounding volume hierarchy, which is the high-level construct we use to cull back the number of ray-box and ray-triangle tests we need to do as illustrated below:



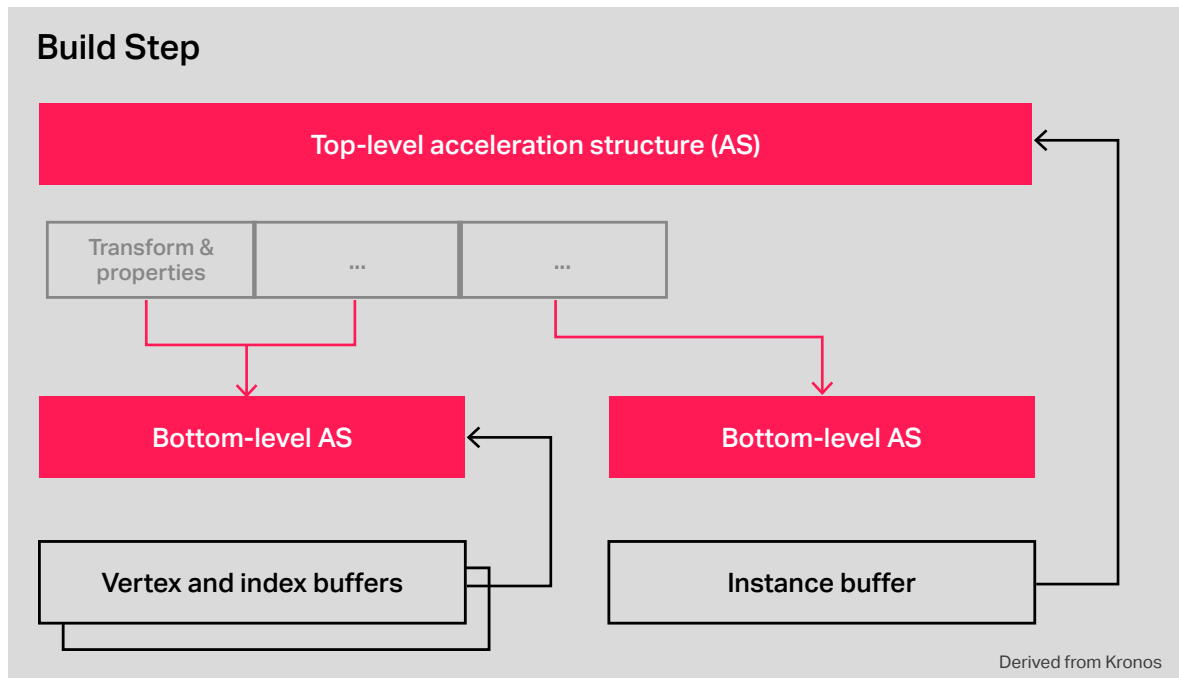
Acceleration structure example. A box/triangle based bounding volume hierarchy enabling hierarchical culling of box/triangle testing work.

As the diagram shows, the bounding volume hierarchy provides an acceleration mechanism where we systematically check the bounding boxes, and if we miss a box, we know we can ignore all the boxes/triangles underneath this level.

This makes it an acceleration structure since we reduce the process of doing the ray testing to the minimum possible. No doubt, many readers will have already realised that this structure, and the quality and heuristics used in its creation, will have a significant impact on how efficient the hardware will be since an optimal structure could cut work away much more effectively than a simplistic, poorly constructed structure. Therefore, the APIs expose both fast and slow methods of generating this acceleration structure.

The fast-building algorithms are essential for objects which are animated and change extensively from frame to frame to maintain high frame rates. The slow-building methods should be used at load time (or even offline during development) for static objects which will be used throughout their lifetime and hence should be as optimal as possible. It's also important to understand what "animated" really means for an acceleration structure and this will become clear as we introduce the next concept.

A Tale of Two Ray Tracing APIs



Acceleration Structure Hierarchy

Acceleration structures are built out of two elements, a top-level acceleration structure (TLAS) and multiple bottom-level acceleration structures (BLAS). What we have described above is more of BLAS, as it contains an acceleration structure for an object, such as the bunny rabbit in our example. The TLAS consists of multiple BLAS structures.

Effectively, the TLAS instances BLAS structures one or many times to build the world, where the same BLAS can have multiple transformation matrices (which move them around in the world) and properties. As such, it's similar to "instancing" used in traditional graphics geometry processing.

What this also means is that moving objects, e.g., translations or rotations of items in the world, are handled only in the TLAS and do not change the BLAS itself. What this means is that a car driving around in the world is a static object and with a static BLAS, the moving of the car and the rotation of its wheels is possible with only modifications of the TLAS transformations. Also, note that for ray tracing the movements of the camera just change the direction and traversal of the rays through the acceleration structure, hence camera changes also do not impact the BLAS or TLAS information. TLAS updates are very cheap and adding/removing objects and changing transforms is quick and easy. However, building/creating a new BLAS is expensive.

Some readers will have already realised that dynamic BLAS cases are possible in games: notable techniques include matrix-skinned character changes or dynamic morphing distorting objects using shaders. These require rebuilding the BLAS and there is a fast/balanced algorithm in real-time which should be used. Memory footprint and bandwidth are also critical, so instancing objects is key. We do not want to store each tree repeatedly in the structure when we can just instance them, since a tree instanced many times requires storage only once as BLAS (largest data structure) and only requires extra TLAS references. This is very important since referencing the same BLAS means not only less storage cost in system memory it also means potentially

A Tale of Two Ray Tracing APIs

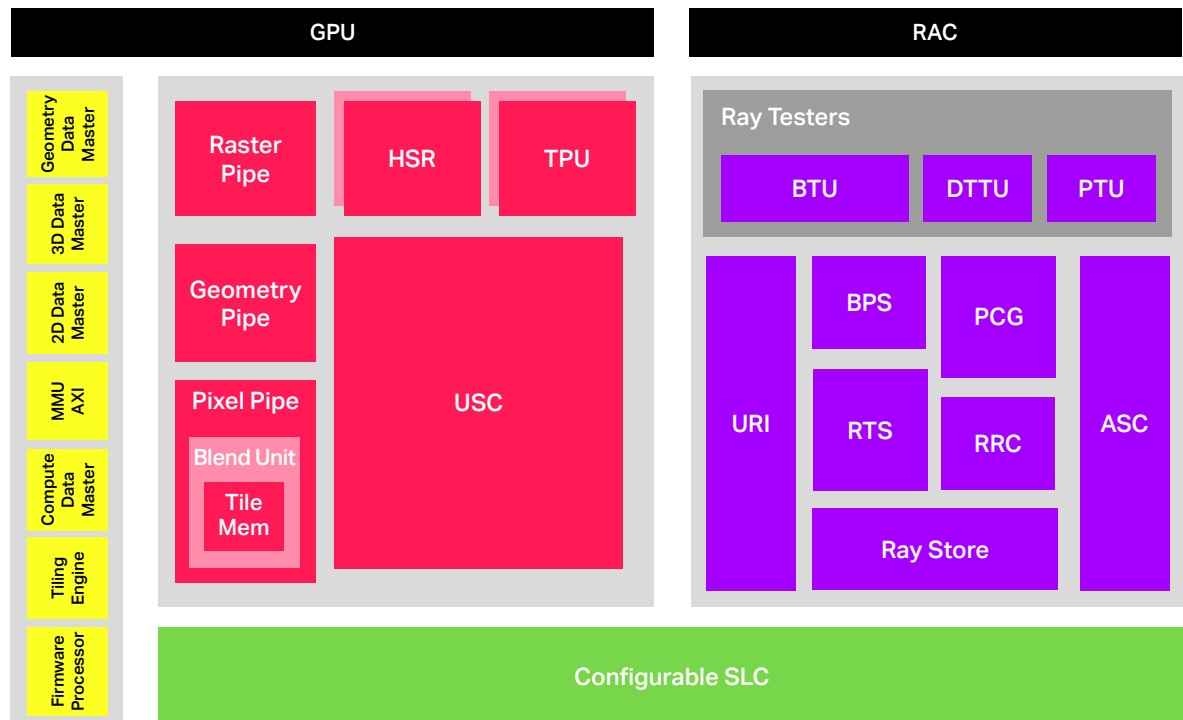
better cache hits and data reuse in traversal since each instance points at the same data in the BLAS.

So, what exactly is inside the BLAS and how is it created? Well, this is opaque to the developer. It's the secret sauce that each vendor brings to the table and the APIs provide TLAS/BLAS building calls in the API with different performance characteristics. Using the correct mode will be important and using offline tools, where available, will also be key for mobile usage.

.

A Walk Through the GPU and RAC

Before we hit the RAC, a variety of other processing steps are needed inside the GPU and for a hybrid rendering workload using ray queries this can be visualised in the below diagram.



The RAC block on the right is tightly integrated with the GPU block on the left

The application renders a scene by issuing API calls, which are processed by our GPU driver constructing command buffers and data structures (textures, shaders, buffers) in memory. The driver will also kick the hardware, potentially waking it up from power-saving modes or just flagging that more work is available for processing. This kick triggers the embedded firmware processor, which will handle all internal activity management and ensure that all jobs respect the priority levels set.

Typically, the first thing which will happen is to kick off geometry processing, meaning that draw calls become tasks inside the GPU, where each task is scheduled and will aim to reserve required resources inside the USC for processing. Vertex/geometry data would then be fetched, and, as data is available, tasks become active and would execute the shader program. This generates output geometry, which then hits a range of fixed-function blocks, such as culling, clipping, tiling and geometry compression, followed by a write-out to memory of the intermediate parameter data.

This parameter data is the per-tile linked list of geometry, which is potentially visible within each tile, thus enabling our tile-based deferred rendering to work its magic. All of this work is the first phase of processing, which we typically refer to as the geometry phase or tile accelerator (TA) phase, a name which you can see in our performance analysis tools. This phase runs concurrently with the next rendering phase.

A Walk Through the GPU and RAC

3D processing within a tile-based deferred rendering architecture starts with hidden surface removal. All 3D processing is done – as the name suggests – tile by tile. This means that position data is fetched using the parameter-data linked lists structures. For all geometry data within the tile depth/stencil tests are executed, generating visibility lists inside a tag buffer, which indicates what objects are visible for each respective pixel. Once all geometry is processed, we thus have per-pixel tagged visibility lists, which logically is one single opaque object (since everything behind it would be hidden/removed) and optionally several alpha-blended layers in front of the opaque object.

Rendering is then kicked off in the correct depth order and sorted per shader with each of these representing a task. Task processing means that first, the schedulers reserve required resources within the USC for processing, then the task and data are prefetched before the task becomes active and the correct shader program instructions are executed. This is where the RAC will be triggered if the shader program in the task contains ray query calls.

For shaders with ray query calls the task will not only request USC resources but also request RAC resources. Executing the actual ray tracing happens as the shader emits the required ray information into the RAC using the USC/RAY Interface (URI) and this information is stored in the Ray Store.

Similar to texture operations, following the transmission of the required ray information into the RAC, the USC will place the task into a de-scheduled wait state, effectively meaning that the USC will start to work on other tasks/jobs while the RAC does its work. As you can imagine, all this work is massively parallel as we will not just be processing one fragment/work item or ray but are processing many threads in parallel within each task (warp). The hardware will also have many such tasks in flight to ensure latency absorption and high utilisation. Effectively then, the RAC will have many rays stored which require processing.

At this point, each ray is tracked by the ray reference counter, which increases for each test required. These start from one and will increment as more boxes are intersected, thus triggering more box tests, as per the acceleration structure. The ray processing is done in coherent groups which means that the packet coherency gathering block will be scanning through rays aiming to build packets of rays that coherently traverse the structure. As packets fill up, they will be executed, running the rays through the box and/or triangle and/or primitive testers as needed. This processing runs via a dedicated acceleration structure cache (ASC), which ensures that data is also re-used across packets.

The ASC is only one cache level of course – further caching will happen throughout the whole GPU cache memory hierarchy, including the largest SLC cache level and potentially even system-level caches at the SoC level. As this processing completes, the Ray Reference Counter (RRC) will increment and decrement as tests are scheduled and completed until processing is finished when the reference count hits 0 and a result is ready for the ray.

At this point the ray, or rays, will be scheduled to return control to the USC for further shader processing and this means that the USC task will be resumed. The resulting ray data can then be read by the USC via the URI from the Ray Store which had reserved resources for all processing.

A Walk Through the GPU and RAC


At this stage processing of shaders would continue as normal until the tile is completely drawn by executing a mix of shaders/kernels with and without ray queries. During this process, other fixed-function blocks such as the texture processing unit would be used to execute the shaders.

It's important to realise the execution at this point is a mix of many tasks: geometry will be processing, compute tasks could be in flight, the RAC would be tracing rays and finding hits/misses, while the shader core is executing code as part of all of these operations. 2D and housekeeping tasks could also be in flight to copy data or generate mipmaps. With such a diverse range of jobs, we aim to obtain maximum efficiency across all processing units and ensure that the latency of any processing tasks and memory access is fully hidden by processing other independent tasks.

Once the tile is finished this would trigger the Pixel Back End, which writes the finished tile to memory, potentially using PowerVR image compression (PVRIC) framebuffer compression.

Developer Tools

The RAC unit is fully instrumented with hardware counters to provide developers with extensive insights into the efficiency of their ray tracing application as it runs. Access to these counters is provided by PVRTune application, which provides insights into a wide range of traditional rasterisation counters as well as a new set of ray-tracing specific counters.



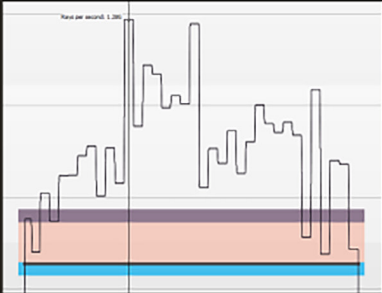
EASY DEPLOYMENT FOR DEVELOPERS

PVRTune now enables developers to see low-level ray tracing counters, such as;

- ▲ Rays per second
- ▲ Box Tester load
- ▲ Cache hit rate
- ▲ Traverse (recursive) rays per second, and many more

New PVRRayTracingSimulation Vulkan Layer in Imagination SDK

Simulate the capabilities and behaviour of our GPUs

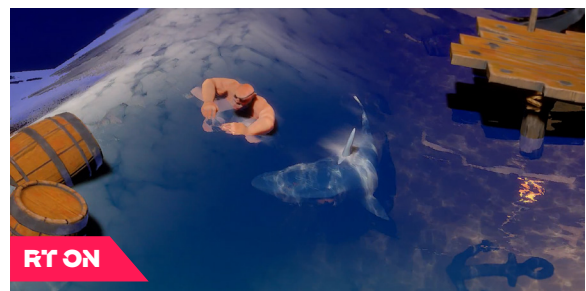


PVRTune aids developers by offering deep insights into ray tracing application behaviour

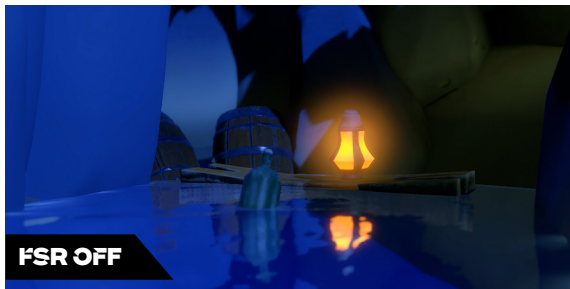
[The latest version of our SDK](#) also includes a PVRRayTracingSimulation. A small collection of Vulkan layers that, as the name suggests, simulates the capabilities and behaviour of PowerVR ray tracing hardware; including acceleration structures. This is an exciting time for graphics development as more and more hardware offers ray tracing support and these tools allow developers to make a head start. The PVRRayTracingSimulation binaries can be downloaded from our developer portal.

IMG DXT Demo

[Our demo](#) highlights the new features and benefits of the IMG DXT GPU using a real-time in-house developed demo. As discussed in this white paper, it demonstrates how Fragment Shading Rate is used to reduce the computational complexity of scenes, and therefore the number of rays needed to generate convincing ray-traced shadows, reflections, and refractions in the demo scene. The demo uses advanced temporal anti-aliasing (TAA) denoising techniques that take advantage of DXT's innovative 2D Dual Rate Texturing Feature, delivering up to double the performance for post-processing effects compared to the previous generation. With FSR combined with 2D Dual Rate Texturing DXT will enable mainstream mass-market deployment of hardware ray tracing building up a viable ecosystem of developers.



RT – Here we see ray tracing adding reflection and refraction effects, adding realism to the scene.



FSR – FSR reduces the number of times the fragment shader is run, and as can be seen, has minimal impact on image quality.



TAA – When the number of rays sent into the scene is reduced (due to modest ray budget counts), visual artefacts are inevitably introduced, which require cleaning up. IMG DXT's 2D Dual-Rate Texturing enhancement improves the efficiency and performance of our TAA denoising solution compared to our previous generation GPU.

Conclusion

In this paper, we have taken you on a journey through Imagination's PowerVR Photon architecture and demonstrated how it has been designed, from the ground up to manage the issues inherent in delivering effective ray tracing in a mobile power budget. We have also shown how with our relentless focus on optimising our GPUs for maximum power and efficiency, we have evolved the Photon architecture to offer lower cost options, to give our customers even more flexibility and choice to enable them to bring ray tracing to the masses, simplifying the creative process for developers and delighting end users with mobile devices capable of delivering incredible visual experiences.

PowerVR TBDR and Coherency Sorting

PowerVR pioneered tile-based deferred rendering (TBDR) in 1996, over 25 years ago. The focus of TBDR is efficiency, both in processing as well as bandwidth. Tile-based rendering does this by sorting all the triangle geometry into screen-space tiled regions first before rendering. This is different from immediate mode rendering (IMR) where every triangle is transformed and immediately drawn. The benefit of sorting all geometry and then rendering per screen-space tile region (usually 16x16 or 32x32 pixels in size), is that we can complete the rendering of the tile region solely using on-chip memory for the depth/stencil buffer as well as the colour buffer. IMRs push all this bandwidth off-chip and depend on cache hits to reduce it, but as geometry submissions are not spatially coherent in screen space this caching approach typically fails, leading to high bandwidth, latency sensitivity and poor power efficiency.

Therefore, by sorting geometry first the cache hit rate effectively becomes 100%. Additionally, depth and stencil buffers are often only used once and hence can be discarded. With GBuffer and MRT rendering many of the MRT "colour" targets are only used for intermediate scratchpad data and only one colour buffer is required to be written out to memory. With TBDR, all of this can be done on-chip, saving memory footprint and very significant amounts of bandwidth.

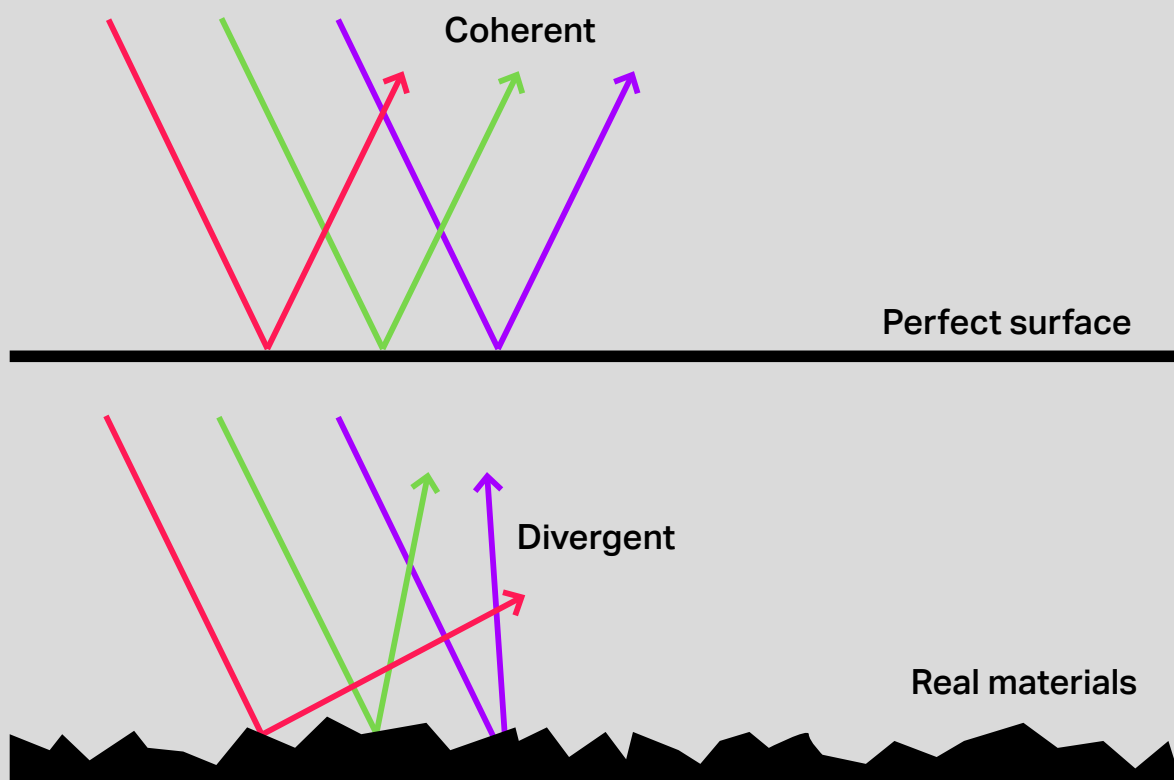
TBDR also offer significant benefits in handling anti-aliasing. As the oversampled buffers only ever exist in on-chip memory, only the downsampled colour targets are written out to memory and the framebuffer, yet again saving memory footprint and bandwidth.

The PowerVR Photon ray tracing architecture is in many ways identical to the PowerVR TBDR architecture in that a spatial sort is also done, only rather than in 2D screen space we bin rays into packets which travel along similar paths through the BVH. The benefits here are similar to what we find with coherency sorting; namely significant cache efficiency and reduced bandwidth, while processing remains in a SIMD/SIMT nature, ensuring high power efficiency of the logic and overall processing.

Conclusion

Hidden Coherency When Ray Tracing

While ray tracing is “embarrassingly parallel” in nature, one of the reasons why real-time ray tracing has taken so long to become practical is that the parallelism is there but it’s very often divergent and non-coherent. This can be understood from the below illustration.



When rays hit surfaces that are not perfectly smooth, they will bounce off in different directions, causing divergence.

In the real world, materials have different properties – some are smooth, but most are rough – and therefore, for realistic surfaces, rays will not be reflected in the same way, but rather bounce in a variety of directions. This results in divergence, e.g. the ray bounces from one pixel to the next pixel resulting in rays going in very different directions. Consequently, the ray will cross the BVH boxes along different paths – thus causing divergent memory accesses – and, logically, rays travelling in different directions will also intersect with different triangles, triggering different shader programs – thus causing divergence in the shader execution.

Divergence is bad for GPUs, as while they are great at processing highly parallel workloads their SIMD architectures only makes sense if those workloads are coherent and similar. If each pixel wants to do something different, the tricks upon which GPUs depend for high execution and bandwidth efficiency fail. This means you end up with a brute force approach (i.e., the use of large amounts of ALUs and ray tracing units), which is required to compensate as the processing flow struggles to use them efficiently (namely despite high peak throughput count on paper, poor utilisation delivers low throughput numbers in real-world use).

Conclusion

Now, while rays from one pixel to the next may be divergent this does not mean that there is no “coherency” among the soup of rays that are bouncing around. Again, this is best illustrated in the image on the next page.

The reflective shape below shows hidden coherency in the rays, which reflect from this object e.g., you can see that the person wearing yellow is reflected many times, meaning those rays go into the same direction and are, indeed, coherent. What’s more, if we can group those rays, they will follow a similar path through the BVH, providing us with a high rate of cache hits and data re-use. They will also ultimately hit and intersect with the same triangles and would likely also execute the same or similar shader programs, consequently delivering high efficiency in traditional parallel GPU ALU pipelines.

What we need therefore is a way of capturing this “hidden” coherency to deliver this efficiency improvement. Imagination did so with its 2014 PowerVR Wizard GPU architecture, which pioneered real-time ray tracing within a modern GPU architecture and introduced concepts such as hybrid rendering (mixing traditional and ray-traced rendering), by including a coherency sorting engine.



The Packet Coherency Gather engine finds and sorts coherent rays in a scene and then packages them up for efficient processing on the GPU.



www.imaginationtech.com

[Contact us now](#)