NVIDIA TURING GPU ARCHITECTURE

Graphics Reinvented

WP-09183-001_v01

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION TO THE NVIDIA TURING ARCHITECTURE

Fueled by the ongoing growth of the gaming market and its insatiable demand for better 3D graphics, NVIDIA® has evolved the GPU into the world's leading parallel processing engine for many computationally-intensive applications. In addition to rendering highly realistic and immersive 3D games, NVIDIA GPUs also accelerate content creation workflows, high performance computing (HPC) and datacenter applications, and numerous artificial intelligence systems and applications.

Turing represents the biggest architectural leap forward in over a decade, providing a new core GPU architecture that enables major advances in efficiency and performance for PC gaming, professional graphics applications, and deep learning inferencing.

Using new hardware-based accelerators and a Hybrid Rendering approach, Turing fuses rasterization, real-time ray tracing, AI, and simulation to enable incredible realism in PC games, amazing new effects powered by neural networks, cinematic-quality interactive experiences, and fluid interactivity when creating or navigating complex 3D models.

Within the core architecture, the key enablers for Turing's significant boost in graphics performance are a new GPU processor (streaming multiprocessor—SM) architecture with improved shader execution efficiency, and a new memory system architecture that includes support for the latest GDDR6 memory technology.

Image processing applications such as the ImageNet Challenge were among the first success stories for deep learning, so it is no surprise that AI has the potential to solve many important problems in graphics. Turing's Tensor Cores power a suite of new deep learning-based *Neural Services* that offer stunning graphics effects for games and professional graphics, in addition to providing fast AI inferencing for cloud-based systems.

The long-sought after holy-grail of computer graphics rendering—real-time ray tracing—is now reality in single-GPU systems with the NVIDIA Turing GPU architecture. Turing GPUs introduce new RT Cores, accelerator units that are dedicated to performing ray tracing operations with extraordinary efficiency, eliminating expensive software emulation-based ray tracing approaches of the past. These new units, combined with NVIDIA RTX™ software technology and sophisticated filtering algorithms, enable Turing to deliver real-time ray-traced rendering, including photorealistic objects and environments with physically accurate shadows, reflections, and refractions.

In parallel with Turing's development, Microsoft announced both the DirectML for AI and DirectX Raytracing (DXR) APIs in early 2018. With the combination of Turing GPU architecture and the new AI and ray tracing APIs from Microsoft, game developers can rapidly deploy real-time AI and ray tracing in their games.

In addition to its groundbreaking AI and ray tracing features, Turing also includes many new advanced shading features that improve performance, enhance image quality, and deliver new levels of geometric complexity.

Turing GPUs also inherit all the enhancements to the NVIDIA CUDA™ platform introduced in the Volta architecture that improve the capability, flexibility, productivity, and portability of compute applications. Features such as independent thread scheduling, hardware-accelerated Multi Process Service (MPS) with address space isolation for multiple applications, and Cooperative Groups are all part of the Turing GPU architecture.

Several of the new NVIDIA GeForce® and NVIDIA Quadro™ GPU products will be powered by Turing GPUs. In this paper we focus on the architecture and capabilities of NVIDIA's flagship Turing GPU, which is codenamed TU102 and will be shipping in the GeForce RTX 2080 Ti and Quadro RTX 6000. Technical details, including product specifications for TU104 and TU106 Turing GPUs, are located in the appendices.

Figure 1 shows how Turing reinvents graphics with an entirely new architecture that includes enhanced Tensor Cores, new RT Cores, and many new advanced shading features. Turing combines programmable shading, real-time ray tracing, and AI algorithms to deliver incredibly realistic and physically accurate graphics for games and professional applications.

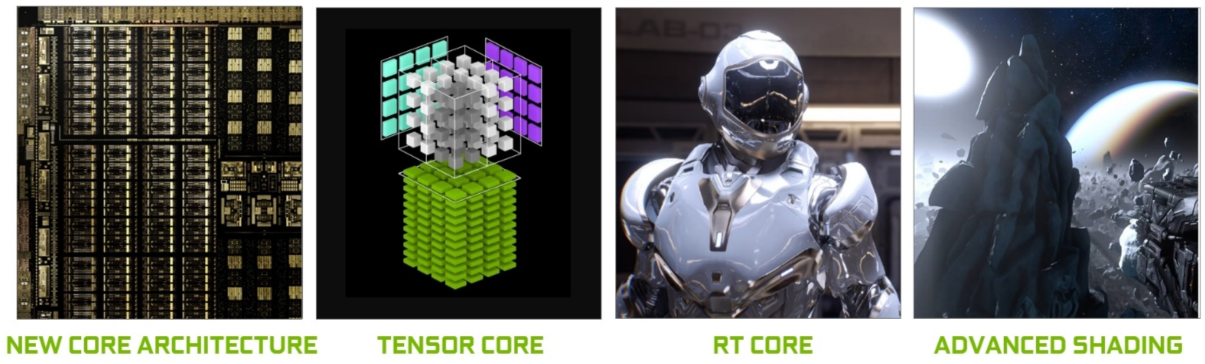


Figure 1. Turing Reinvents Graphics

NVIDIA TURING KEY FEATURES

NVIDIA Turing is the world's most advanced GPU architecture. The high-end TU102 GPU includes 18.6 billion transistors fabricated on TSMC's 12 nm FFN (FinFET NVIDIA) high-performance manufacturing process.

The GeForce RTX 2080 Ti Founders Edition GPU delivers the following exceptional computational performance:

- 14.2 TFLOPS[1] of peak single precision (FP32) performance
- 28.5 TFLOPS[1] of peak half precision (FP16) performance
- 14.2 TIPS[1] concurrent with FP, through independent integer execution units
- 113.8 Tensor TFLOPS[1,2]
- 10 Giga Rays/sec
- 78 Tera RTX-OPS[3]

The Quadro RTX 6000 provides superior computational performance designed for professional workflows:

- 16.3 TFLOPS[1] of peak single precision (FP32) performance
- 32.6 TFLOPS[1] of peak half precision (FP16) performance
- 16.3 TIPS[1] concurrent with FP, through independent integer execution units
- 130.5 Tensor TFLOPS[1,2]
- 10 Giga Rays/sec
- 84 Tera RTX-OPS[3]

The following section describes Turing's major new innovations in summary format. More detailed descriptions of each area are provided throughout this whitepaper.

New Streaming Multiprocessor (SM)

Turing introduces a new processor architecture, the **Turing SM**, that delivers a dramatic boost in shading efficiency, achieving 50% improvement in delivered performance per CUDA Core compared to the Pascal generation. These improvements are enabled by two key architectural changes. First, the Turing SM adds a new independent integer datapath that can execute instructions concurrently with the floating-point math datapath. In previous generations, executing these instructions would have blocked floating-point instructions from issuing. Second, the SM memory path has been redesigned to unify shared memory, texture caching, and memory load caching into one unit. This translates to 2x more bandwidth and more than 2x more capacity available for L1 cache for common workloads.

[1] Based on GPU Boost clock.

[2] FP16 matrix math with FP16 accumulation.

[3] See *Appendix C*
RTX-OPS Description for RTX-OPS details.

Turing Tensor Cores

Tensor Cores are specialized execution units designed specifically for performing the tensor / matrix operations that are the core compute function used in Deep Learning. Similar to Volta Tensor Cores, the Turing Tensor Cores provide tremendous speed-ups for matrix computations at the heart of deep learning neural network training and inferencing operations. Turing GPUs include a new version of the Tensor Core design that has been enhanced for inferencing. Turing Tensor Cores add new INT8 and INT4 precision modes for inferencing workloads that can tolerate quantization and don't require FP16 precision. Turing Tensor Cores bring new deep learning-based AI capabilities to GeForce gaming PCs and Quadro-based workstations for the first time. A new technique called Deep Learning Super Sampling (DLSS) is powered by Tensor Cores. DLSS leverages a deep neural network to extract multidimensional features of the rendered scene and intelligently combine details from multiple frames to construct a high-quality final image. DLSS uses fewer input samples than traditional techniques such as TAA, while avoiding the algorithmic difficulties such techniques face with transparency and other complex scene elements.

Real-Time Ray Tracing Acceleration

Turing introduces real-time ray tracing that enables a single GPU to render visually realistic 3D games and complex professional models with physically accurate shadows, reflections, and refractions. Turing's new RT Cores accelerate ray tracing and are leveraged by systems and interfaces such as NVIDIA's RTX ray tracing technology, and APIs such as Microsoft DXR, NVIDIA OptiX™, and Vulkan ray tracing to deliver a real-time ray tracing experience.

New Shading Advancements

Mesh Shading

Mesh shading advances NVIDIA's geometry processing architecture by offering a new shader model for the vertex, tessellation, and geometry shading stages of the graphics pipeline, supporting more flexible and efficient approaches for computation of geometry. This more flexible model makes it possible, for example, to support an order of magnitude more objects per scene, by moving the key performance bottleneck of object list processing off of the CPU and into highly parallel GPU mesh shading programs. Mesh shading also enables new algorithms for advanced geometric synthesis and object LOD management.

Variable Rate Shading (VRS)

VRS allows developers to control shading rate dynamically, shading as little as once per sixteen pixels or as often as eight times per pixel. The application specifies shading rate using a combination of a shading-rate surface and a per-primitive (triangle) value. VRS is a very powerful tool that allows developers to shade more efficiently, reducing work in regions of the screen where full resolution shading would not give any visible image quality benefit, and therefore improving frame rate. Several classes of VRS-based algorithms have already been identified, which can vary shading work based on content level of detail (Content Adaptive Shading), rate of content motion (Motion Adaptive Shading), and for VR applications, lens resolution and eye position (Foveated Rendering).

Texture-Space Shading

With texture-space shading, objects are shaded in a private coordinate space (a texture space) that is saved to memory, and pixel shaders sample from that space rather than evaluating results directly. With the ability to cache shading results in memory and reuse/resample them, developers can eliminate duplicate shading work or use different sampling approaches that improve quality.

Multi-View Rendering (MVR)

MVR powerfully extends Pascal's Single Pass Stereo (SPS). While SPS allowed rendering of two views that were common except for an X offset, MVR allows rendering of multiple views in a single pass even if the views are based on totally different origin positions or view directions. Access is via a simple programming model in which the compiler automatically factors out view independent code, while identifying view-dependent attributes for optimal execution.

Deep Learning Features for Graphics

NVIDIA NGX™ is the new deep learning-based neural graphics framework of NVIDIA RTX Technology. NVIDIA NGX utilizes deep neural networks (DNNs) and set of "Neural Services" to perform AI-based functions that accelerate and enhance graphics, rendering, and other client-side applications. NGX employs the Turing Tensor Cores for deep learning-based operations and accelerates delivery of NVIDIA deep learning research directly to the end-user. Features include ultra-high quality NGX DLSS (Deep Learning Super-Sampling), AI InPainting content-aware image replacement, AI Slow-Mo very high-quality and smooth slow motion, and AI Super Rez smart resolution resizing.

Deep Learning Features for Inference

Turing GPUs deliver exceptional inference performance. The Turing Tensor Cores, along with continual improvements in TensorRT (NVIDIA's run-time inferencing framework), CUDA, and CuDNN libraries, enable Turing GPUs to deliver outstanding performance for inferencing applications. Turing Tensor Cores also add support for fast INT8 matrix operations to significantly accelerate inference throughput with minimal loss in accuracy. New low-precision INT4 matrix operations are now possible with Turing Tensor Cores and will enable research and development into sub 8-bit neural networks.

GDDR6 High-Performance Memory Subsystem

Turing is the first GPU architecture to support GDDR6 memory. GDDR6 is the next big advance in high-bandwidth GDDR DRAM memory design. GDDR6 memory interface circuits in Turing GPUs have been completely redesigned for speed, power efficiency and noise reduction, achieving 14 Gbps transfer rates at 20% improved power efficiency compared to GDDR5X memory used in Pascal GPUs.

Second-Generation NVIDIA NVLink

Turing TU102 and TU104 GPUs incorporate NVIDIA's NVLink™ high-speed interconnect to provide dependable, high bandwidth and low latency connectivity between pairs of Turing GPUs. With up to 100GB/sec of bidirectional bandwidth, NVLink makes it possible for customized workloads to efficiently split across two GPUs and share memory capacity. For gaming workloads, NVLink's increased bandwidth and dedicated inter-GPU channel enables new possibilities for SLI, such as new modes or higher resolution display configurations. For large memory workloads, including professional ray tracing applications, scene data can be split across the frame buffer of both GPUs, offering up to 96 GB of shared frame buffer memory (two 48 GB Quadro RTX 8000 GPUs), and memory requests are automatically routed by hardware to the correct GPU based on the location of the memory allocation.

USB-C and VirtualLink

Turing GPUs include hardware support for USB Type-C™ and VirtualLink™[4]. VirtualLink is a new open industry standard being developed to meet the power, display, and bandwidth demands of next-generation VR headsets through a single USB-C connector. In addition to easing the setup hassles present in today's VR headsets, VirtualLink will bring VR to more devices.

[4] In preparation for the emerging VirtualLink standard, Turing GPUs have implemented hardware support according to the *VirtualLink Advance Overview*. To learn more about VirtualLink, refer to http://www.virtuallink.org.

TURING GPU ARCHITECTURE IN-DEPTH

The Turing TU102 GPU is the highest performing GPU of the Turing GPU line and the focus of this section. The TU104 and TU106 GPUs utilize the same basic architecture as TU102, scaled down to different degrees for different usage models and market segments. Details of TU104 and TU106 chip architectures and target usages/markets are provided in *Appendix A, Turing TU104 GPU* and *Appendix B, Turing TU106 GPU*.

TURING TU102 GPU

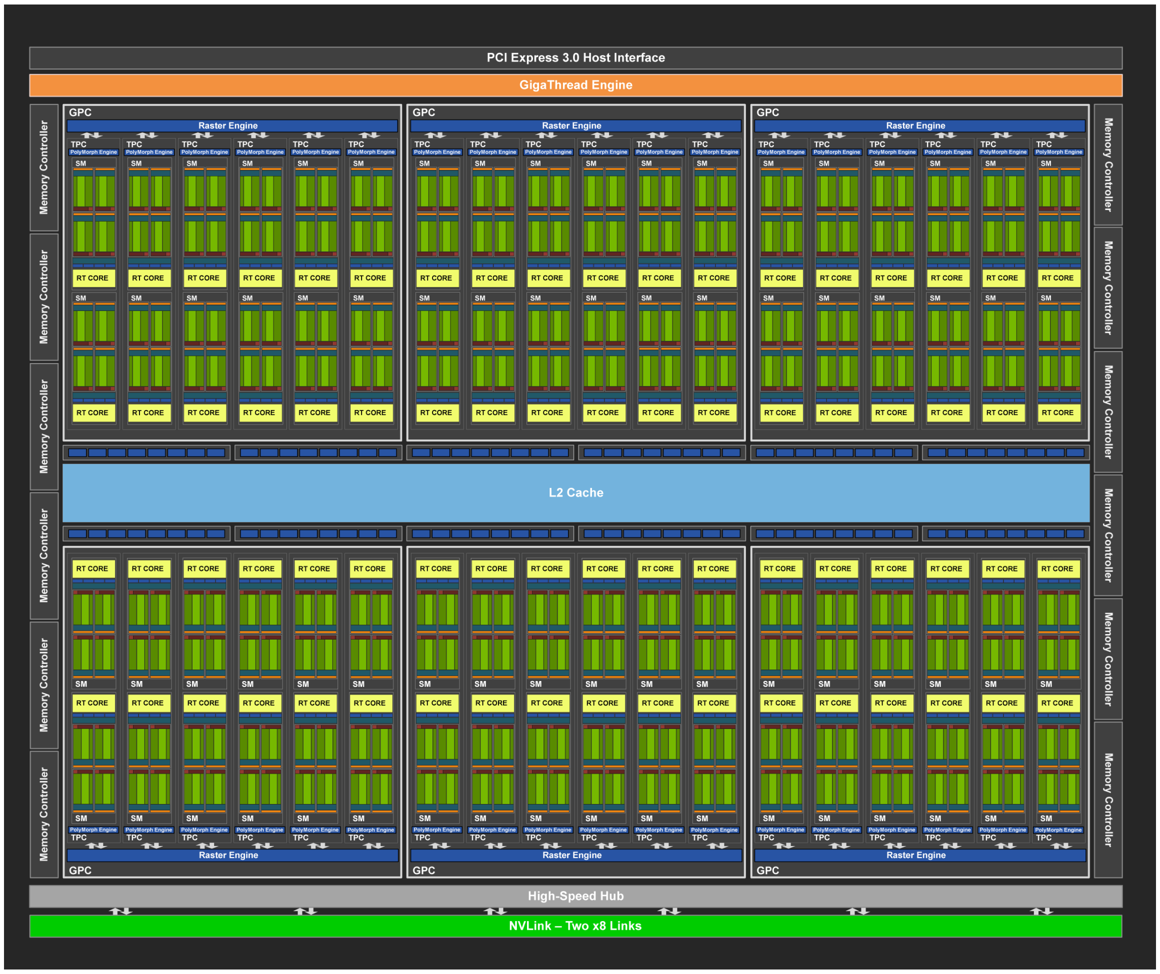
The TU102 GPU includes six Graphics Processing Clusters (GPCs), 36 Texture Processing Clusters (TPCs), and 72 Streaming Multiprocessors (SMs). (See Figure 2 for an illustration of the TU102 full GPU with 72 SM units.) Each GPC includes a dedicated raster engine and six TPCs, with each TPC including two SMs. Each SM contains 64 CUDA Cores, eight Tensor Cores, a 256 KB register file, four texture units, and 96 KB of L1/shared memory which can be configured for various capacities depending on the compute or graphics workloads.

Ray tracing acceleration is performed by a new RT Core processing engine within each SM (RT Core and ray tracing features are discussed in more depth in *Turing Ray Tracing Technology* starting on page 25).

The full implementation of the TU102 GPU includes the following:

- 4,608 CUDA Cores
- 72 RT Cores
- 576 Tensor Cores
- 288 texture units
- 12 32-bit GDDR6 memory controllers (384-bits total).

Tied to each memory controller are eight ROP units and 512 KB of L2 cache. The full TU102 GPU consists of 96 ROP units and 6144 KB of L2 cache. See the Turing TU102 GPU in Figure 3. Table 1 compares the GPU features of the Pascal GP102 to the Turing TU102.



Note: The TU102 GPU also features 144 FP64 units (two per SM), which are not depicted in this diagram. The FP64 TFLOP rate is 1/32nd the TFLOP rate of FP32 operations. The small number of FP64 hardware units are included to ensure any programs with FP64 code operates correctly.

Figure 2. Turing TU102 Full GPU with 72 SM Units

Table 1. Comparison of NVIDIA Pascal GP102 and Turing TU102

GPU Features	**GTX 1080Ti**	**RTX 2080 Ti**	**Quadro P6000**	**Quadro RTX 6000**
Architecture	Pascal	Turing	Pascal	Turing
GPCs	6	6	6	6
TPCs	28	34	30	36
SMs	28	68	30	72
CUDA Cores / SM	128	64	128	64
CUDA Cores / GPU	3584	4352	3840	4608
Tensor Cores / SM	NA	8	NA	8
Tensor Cores / GPU	NA	544	NA	576
RT Cores	NA	68	NA	72
GPU Base Clock MHz (Reference / Founders Edition)	1480 / 1480	1350 / 1350	1506	1455

GPU Features	GTX 1080Ti	RTX 2080 Ti	Quadro P6000	Quadro RTX 6000
GPU Boost Clock MHz (Reference / Founders Edition)	1582 / 1582	1545 / 1635	1645	1770
RTX-OPS (Tera-OPS) (Reference / Founders Edition)	11.3 / 11.3	76 / 78	NA	84
Rays Cast (Giga Rays/sec) (Reference / Founders Edition)	1.1 / 1.1	10 / 10	NA	10
Peak FP32 TFLOPS✱ (Reference/Founders Edition)	11.3 / 11.3	13.4 / 14.2	12.6	16.3
Peak INT32 TIPS✱ (Reference/Founders Edition)	NA	13.4 / 14.2	NA	16.3
Peak FP16 TFLOPS✱ (Reference/Founders Edition)	NA	26.9 / 28.5	NA	32.6
Peak FP16 Tensor TFLOPS with FP16 Accumulate✱ (Reference/Founders Edition)	NA	107.6 / 113.8	NA	130.5
Peak FP16 Tensor TFLOPS with FP32 Accumulate✱ (Reference/Founders Edition)	NA	53.8 / 56.9	NA	130.5
Peak INT8 Tensor TOPS✱ (Reference/Founders Edition)	NA	215.2 / 227.7	NA	261.0
Peak INT4 Tensor TOPS✱ (Reference/Founders Edition)	NA	430.3 / 455.4	NA	522.0
Frame Buffer Memory Size and Type	11264 MB GDDR5X	11264 MB GDDR6	24576 MB GDDR5X	24576 MB GDDR6
Memory Interface	352-bit	352-bit	384-bit	384-bit
Memory Clock (Data Rate)	11 Gbps	14 Gbps	9 Gbps	14 Gbps
Memory Bandwidth (GB/sec)	484	616	432	672
ROPs	88	88	96	96
Texture Units	224	272	240	288
Texel Fill-rate (Gigatexels/sec)	354.4 / 354.4	420.2 / 444.7	395	510
L2 Cache Size	2816 KB	5632 KB	3072 KB	6144 KB
Register File Size/SM	256 KB	256 KB	256 KB	256 KB
Register File Size/GPU	7168 KB	17408 KB	7680 KB	18432 KB
TDP⋆ (Reference/Founders Edition)	250 / 250 W	250 / 260 W	250 W	260 W
Transistor Count	12 Billion	18.6 Billion	12 Billion	18.6 Billion
Die Size	471	754	471	754
Manufacturing Process	16 nm	12 nm FFN	16 nm	12 nm FFN
Note: ✱ Peak TFLOPS, TIPS, and TOPS rates are based on GPU Boost Clock. ⋆ Power figure represents Graphics Card TDP only. Note that use of the VirtualLink™/USB Type-C™ connector requires up to an additional 35 W of power that is not represented in this power figure.				

As GPU-accelerated computing has become more popular, systems with multiple GPUs are increasingly being deployed in servers, workstations, and supercomputers. The TU102 and TU104 GPUs include the second generation of NVIDIA's NVLink™ high-speed interconnect, originally designed into the Volta GV100 GPU, providing high-speed multi-GPU connectivity for SLI and other multi-GPU use cases. NVLink permits each GPU to directly access memory of other connected GPUs, providing much faster GPU-to-GPU communications, and allows combining memory from multiple GPUs to support much larger datasets and faster in-memory computations.

TU102 includes two NVLink x8 links each capable of delivering up to 25 Gigabytes/second in each direction, for a total aggregate bidirectional bandwidth of 100 Gigabytes/second (see Figure 3).

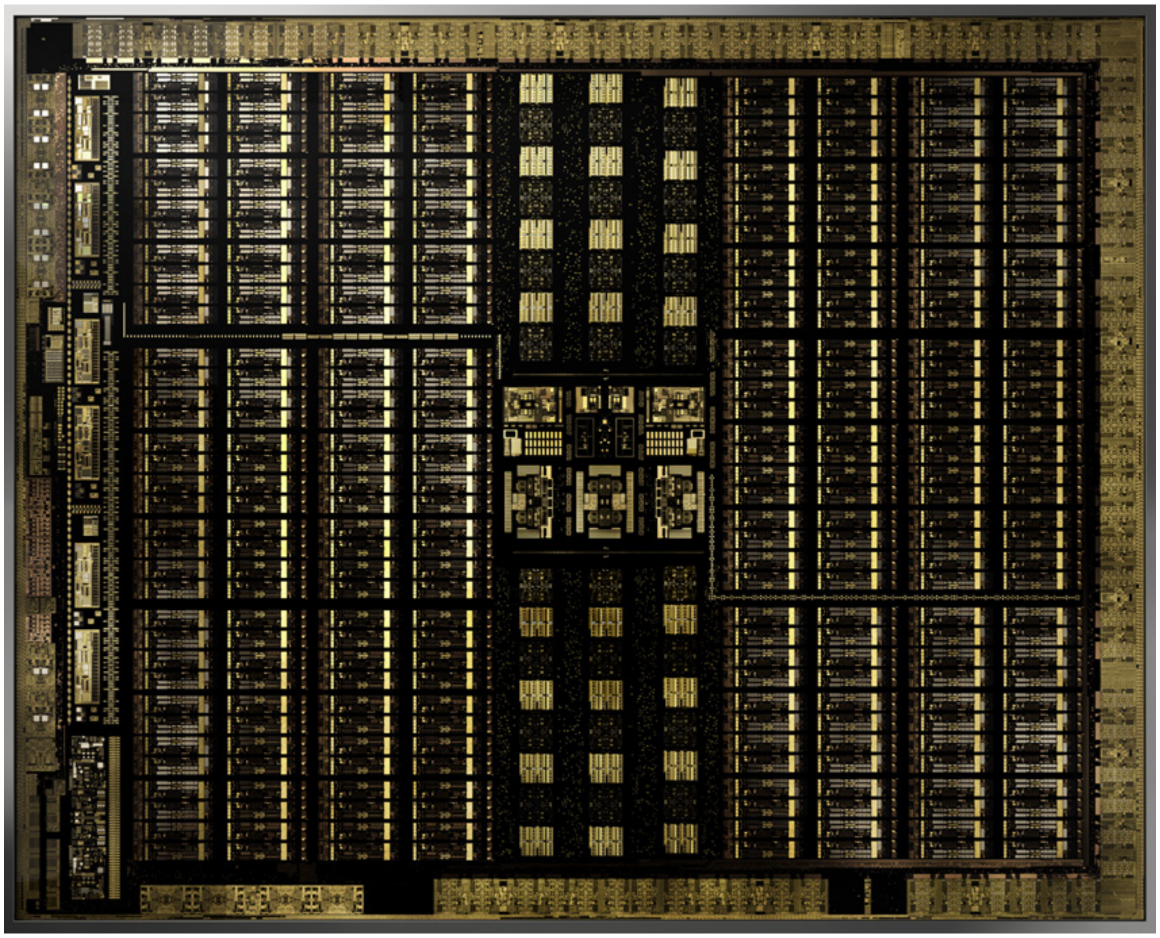


Figure 3. NVIDIA Turing TU102 GPU

TURING STREAMING MULTIPROCESSOR (SM) ARCHITECTURE

The Turing architecture features a new SM design that incorporates many of the features introduced in our Volta GV100 SM architecture. Two SMs are included per TPC, and each SM has a total of 64 FP32 Cores and 64 INT32 Cores. In comparison, the Pascal GP10x GPUs have one SM per TPC and 128 FP32 Cores per SM. The Turing SM supports concurrent execution of FP32 and INT32 operations (more details below), independent thread scheduling similar to the Volta GV100 GPU. Each Turing SM also includes eight mixed-precision Turing Tensor Cores, which are described in more detail in the *Turing Tensor Cores* section on page 15 , and one RT Core, whose functionality is described in the *Turing Ray Tracing Technology section* starting on page 30. See Figure 4 for an illustration of the Turing TU102, TU104, and TU106 SM.

The Turing SM is partitioned into four processing blocks, each with 16 FP32 Cores, 16 INT32 Cores, two Tensor Cores, one warp scheduler, and one dispatch unit. Each block includes a new L0 instruction cache and a 64 KB register file. The four processing blocks share a combined 96 KB L1 data cache/shared memory. Traditional graphics workloads partition the 96 KB L1/shared memory as 64 KB of dedicated graphics shader RAM and 32 KB for texture cache and register file spill area. Compute workloads can divide the 96 KB into 32 KB shared memory and 64 KB L1 cache, or 64 KB shared memory and 32 KB L1 cache.

Turing implements a major revamping of the core execution datapaths. Modern shader workloads typically have a mix of FP arithmetic instructions such as FADD or FMAD with simpler instructions such as integer adds for addressing and fetching data, floating point compare or min/max for processing results, etc. In previous shader architectures, the floating-point math datapath sits idle whenever one of these non-FP-math instructions runs. Turing adds a second parallel execution unit next to every CUDA core that executes these instructions in parallel with floating point math.

Figure 5 shows that the mix of integer pipe versus floating point instructions varies, but across several modern applications, we typically see about 36 additional integer pipe instructions for every 100 floating point instructions. Moving these instructions to a separate pipe translates to an effective 36% additional throughput possible for floating point.

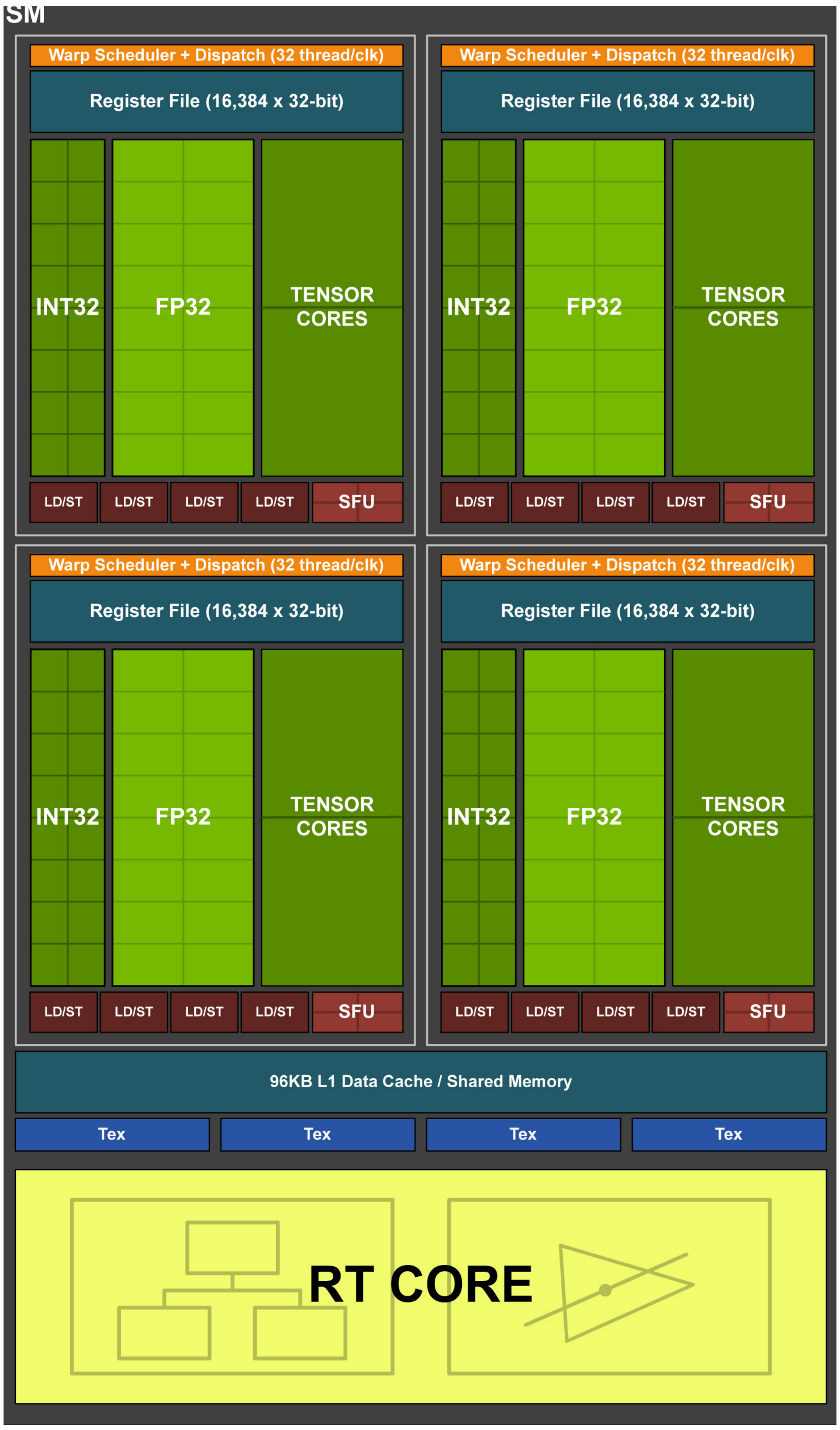
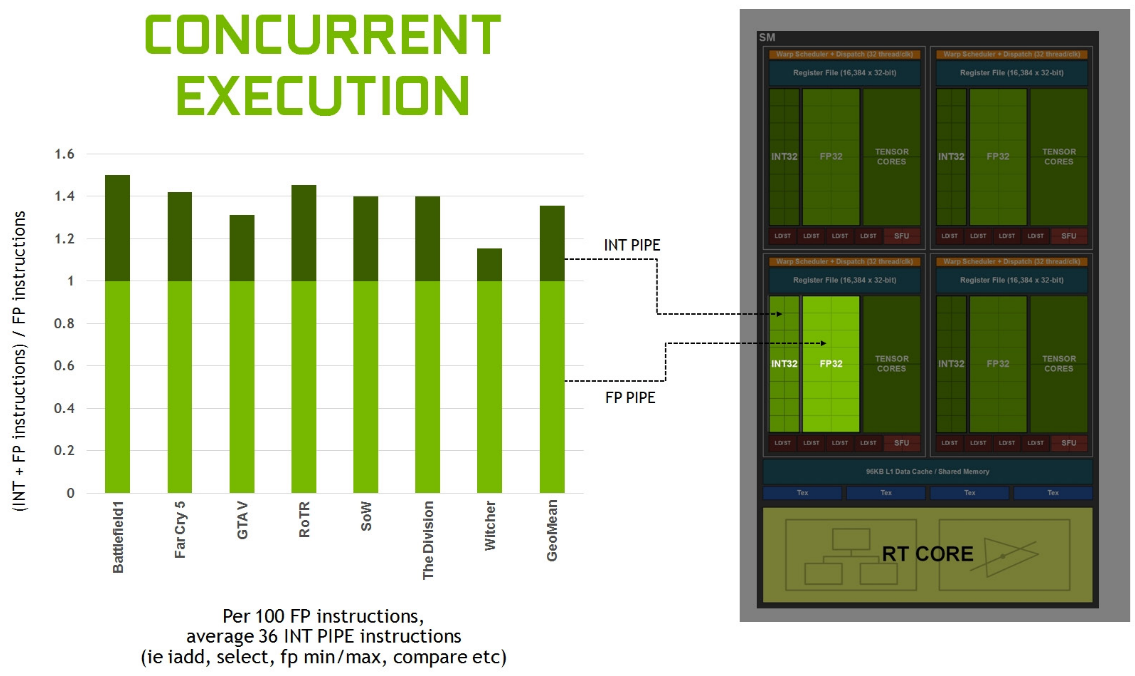

SM
Warp Scheduler + Dispatch (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
TENSOR CORES
LD/ST
SFU
96KB L1 Data Cache / Shared Memory
Tex
RT CORE

Figure 4. Turing TU102/TU104/TU106 Streaming Multiprocessor (SM)



Profiling many workloads shows an average of 36 integer operations for every 100 floating point operations.

Figure 5. Concurrent Execution of Floating Point and Integer Instructions in the Turing SM

Turing's SM also introduces a new unified architecture for shared memory, L1, and texture caching. This unified design allows the L1 cache to leverage resources, increasing its hit bandwidth by 2x per TPC compared to Pascal, and allows it to be reconfigured to grow larger when shared memory allocations are not using all the shared memory capacity. The Turing L1 can be as large as 64 KB in size, combined with a 32 KB per SM shared memory allocation, or it can reduce to 32 KB, allowing 64 KB of allocation to be used for shared memory. Turing's L2 cache capacity has also been increased.

Figure 6 shows how the new combined L1 data cache and shared memory subsystem of the Turing SM significantly improves performance while also simplifying programming and reducing the tuning required to attain at or near-peak application performance. Combining the L1 data cache with the shared memory reduces latency and provides higher bandwidth than the L1 cache implementation used previously in Pascal GPUs.

Overall, the changes in SM enable Turing to achieve 50% improvement in delivered performance per CUDA core. Figure 7 shows the results across a set of shader workloads from current gaming applications.

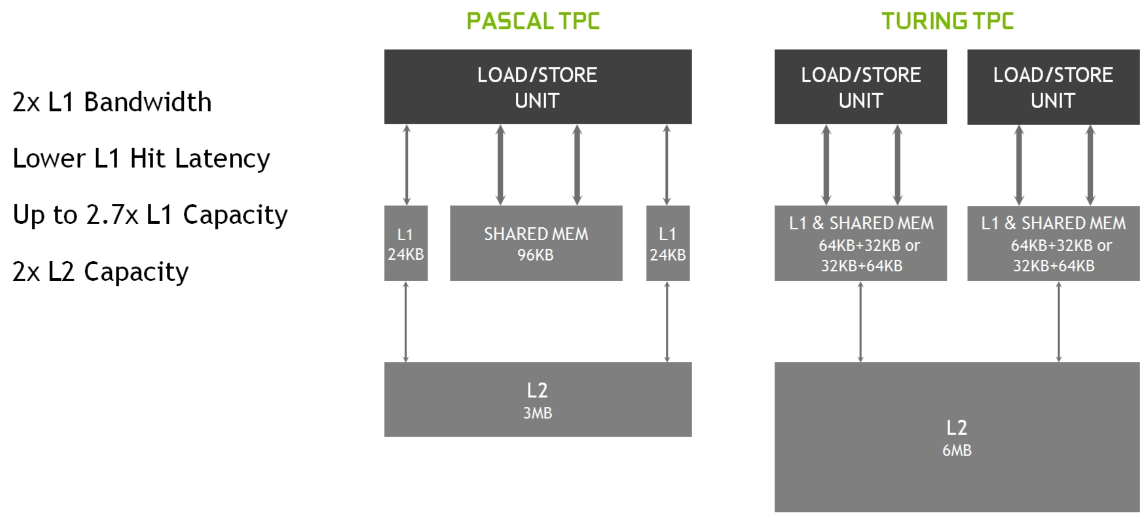


Figure 6. New Shared Memory Architecture

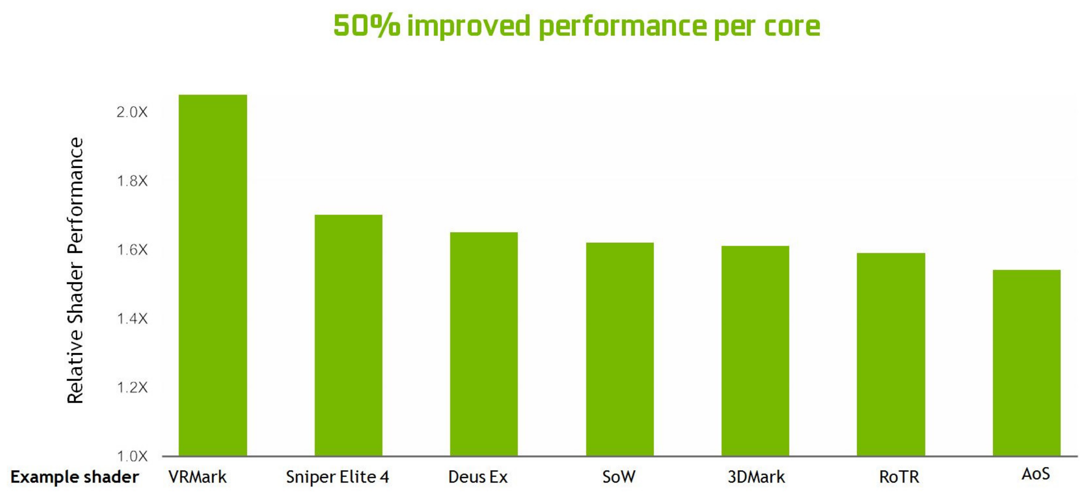


Figure 7. Turing Shading Performance Speedup versus Pascal on Many Different Workloads

Turing Tensor Cores

Turing GPUs include an enhanced version of the Tensor Cores first introduced in the Volta GV100 GPU. The Turing Tensor Core design adds INT8 and INT4 precision modes for inferencing workloads that can tolerate quantization. FP16 is also fully supported for workloads that require higher precision.

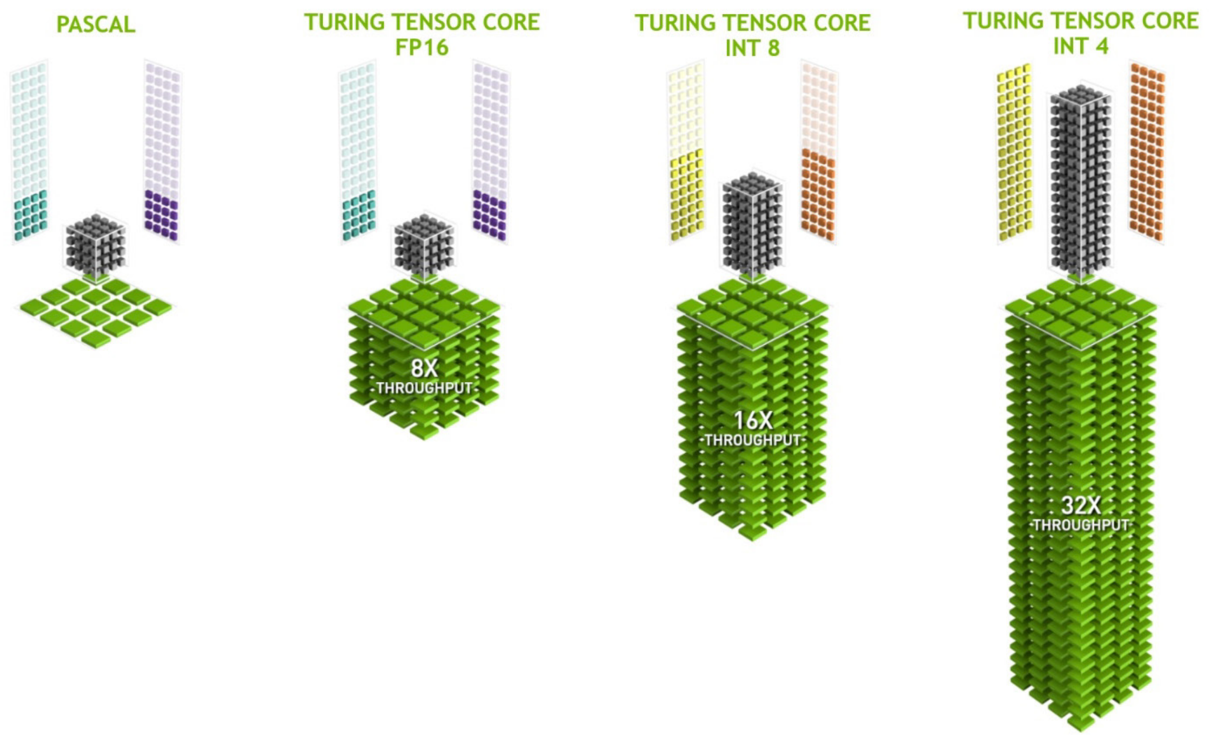
The introduction of Tensor Cores into Turing-based GeForce gaming GPUs makes it possible to bring real-time deep learning to gaming applications for the first time. Turing Tensor Cores accelerate the AI-based features of NVIDIA NGX Neural Services that enhance graphics, rendering, and other types of client-side applications. Examples of NGX AI features include Deep Learning Super Sampling (DLSS), AI InPainting, AI Super Rez, and AI Slow-Mo. More details are included in the *NVIDIA NGX Technology* section on page 33.

Turing Tensor Cores accelerate the matrix-matrix multiplication at the heart of neural network training and inferencing functions. Turing Tensor Cores particularly excel at inference computations, in which useful and relevant information can be inferred and delivered by a trained deep neural network (DNN) based on a given input. Examples of inference include identifying images of friends in Facebook photos, identifying and classifying different types of automobiles, pedestrians, and road hazards in self-driving cars, translating human speech in real-time, and creating personalized user recommendations in online retail and social media systems.

A TU102 GPU contains 576 Tensor Cores: eight per SM and two per each processing block within an SM. Each Tensor Core can perform up to 64 floating point fused multiply-add (FMA) operations per clock using FP16 inputs. Eight Tensor Cores in an SM perform a total of 512 FP16 multiply and accumulate operations per clock, or 1024 total FP operations per clock. The new INT8 precision mode works at double this rate, or 2048 integer operations per clock.

Turing Tensor Cores provide significant speedups to matrix operations and are used for both deep learning training and inference operations in addition to new neural graphics functions. For more information on basic Tensor Core operational details refer to the *NVIDIA Tesla V100 GPU Architecture Whitepaper.*

Figure 8 shows the new Turing Tensor Cores that provide multi-precision for AI inference.



Pascal **Turing Tensor Core**

Figure 8. New Turing Tensor Cores Provide Multi-Precision for AI Inference

TURING OPTIMIZED FOR DATACENTER APPLICATIONS

NVIDIA GPUs have become the standard industry solution for deep learning training, and GPU-based inferencing is gaining traction and is rapidly being adopted. Many of the world's leading enterprises now employ NVIDIA GPUs for running inferencing applications both in the data center and on edge devices. Many enterprises that have traditionally run inferencing applications on CPUs are now switching over to NVIDIA GPUs and getting amazing increases in performance with minimal effort. For example, the NVIDIA Tesla® P4 GPU-based inferencing on the Pascal architecture delivers an industry leading 10X higher inference performance and 25X higher energy efficiency than CPU-based servers in hyperscale data centers[5]. This lead is further extended by the NVIDIA Tesla T4 GPU, the first Turing-based GPU that provides breakthrough performance with flexible multi-precision capabilities, from FP32 to FP16 to INT8, as well as INT4.

[5] Compared to Intel Xeon Gold 6140 using Intel deep learning deployment tool using Resnet-50.

NVIDIA Tesla T4 is the latest and most advanced inferencing solution for hyperscale data centers that deliver universal inference acceleration that spans applications such as image classification and tagging, video analytics, natural language processing, automatic speech recognition, and intelligent search. The breadth of Tesla T4's inferencing capabilities enables it to be used in enterprise solutions and edge devices.

The NVIDIA Tesla T4 GPU includes 2,560 CUDA Cores and 320 Tensor Cores, delivering up to 130 TOPs (Tera Operations per second) of INT8 and up to 260 TOPS of INT4 inferencing performance (see Appendix A, *Turing TU104 GPU* for more Tesla T4 specifications). Compared to CPU-based inferencing, the Tesla T4, powered by the new Turing Tensor Cores, delivers up to 40X higher inference performance[6] (see Figure 9).

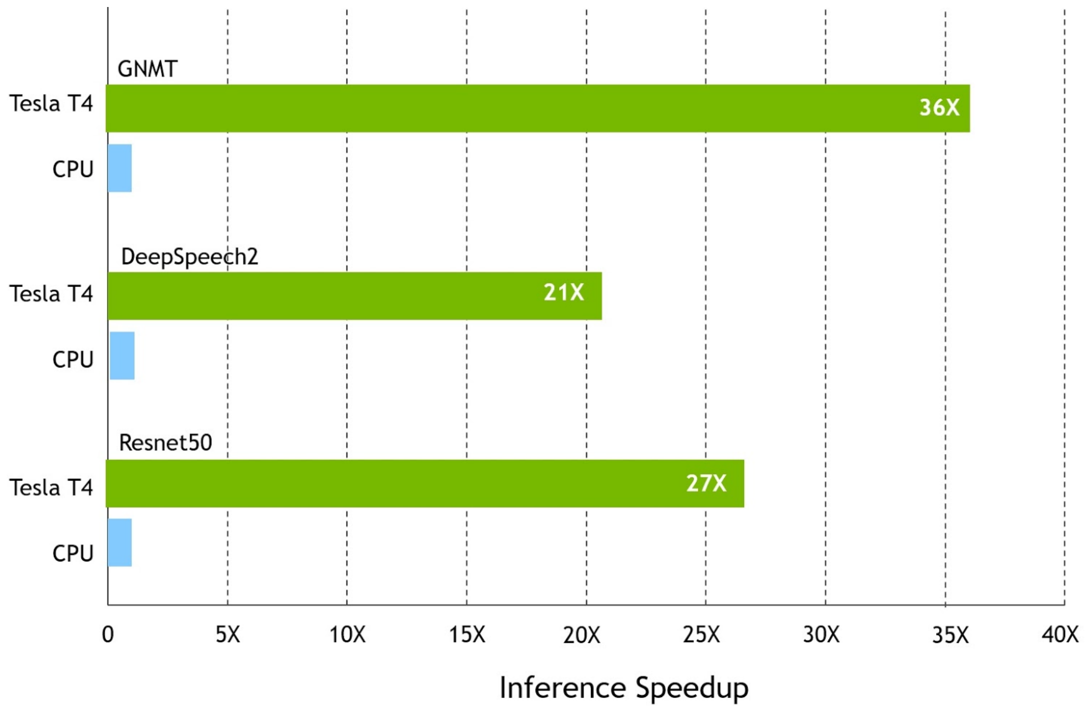


Figure 9. Tesla T4 delivers up to 40X Higher Inference Performance

Energy efficiency is critical for datacenters, and the Tesla T4 delivers more than 50X higher energy efficiency than CPU-based inferencing and up to twice the energy efficiency of NVIDIA's prior generation Tesla P4 GPU[7] (see Figure 10).

[6] Resnet-50 inference throughput at max batch size for latency less than 7 ms. CPU performance measured on Intel Skylake 6140 using Intel OpenVino. GPU performance measured on Tesla T4 using TensorRT5.0.

[7] SKL CPU: Xeon Gold 6140, measured with Intel Deep Learning Deployment Tool, does not achieve 7 ms latency. GPU measured with TensorFlow or TensorRT as mentioned. NVIDIA Tesla T4 performance projections are preliminary and subject to change without notice.

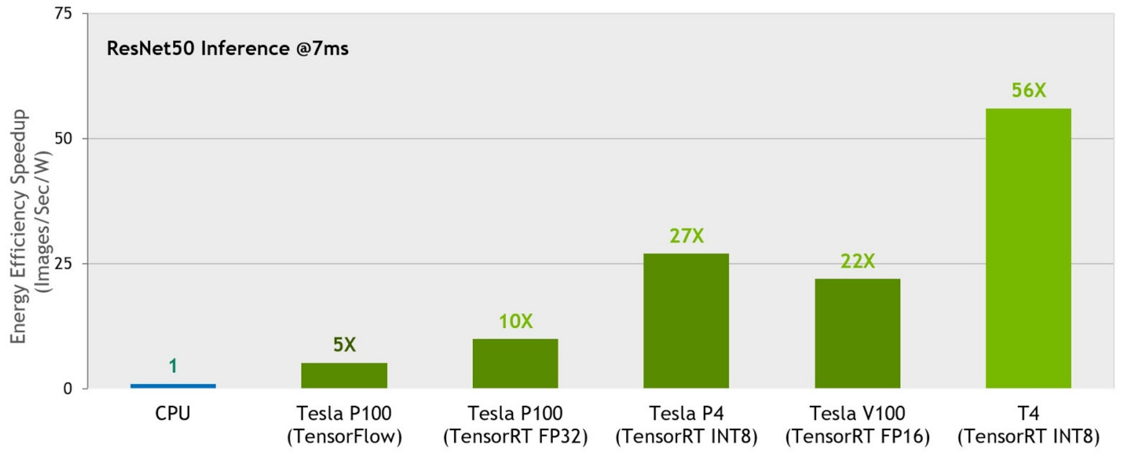


Figure 10. Tesla T4 Delivers More than 50X the Energy Efficiency of CPU-based Inferencing

Turing GPU architecture, in addition to Turing Tensor Cores, includes several features to improve performance of data center applications. Some of the key features are:

- Enhanced Video Engine
  Compared to prior generation Pascal and Volta GPU architectures, Turing supports additional video decode formats such as HEVC 4:4:4 (8/10/12 bit), and VP9 (10/12 bit) (see the *Video and Display Engine* section starting on page 21 for more details). The enhanced video engine in Turing is capable of decoding significantly higher number of concurrent video streams than equivalent Pascal based Tesla GPUs (see Table 2).
- Turing Multi-Process Service
  Turing GPU architecture inherits the enhanced Multi-Process Service (MPS) feature first introduced in the Volta architecture. Compared to Pascal-based Tesla GPUs, MPS on Tesla T4 improves inference performance for small batch sizes, reduces launch latency, improves Quality of Service, and enables servicing higher numbers of concurrent client requests.
- Higher memory bandwidth and larger memory size
  With 16 GB of GPU memory and 320 GB/sec of memory bandwidth, Tesla T4 delivers almost double the memory bandwidth and twice the memory capacity of its predecessor the Tesla P4 GPU. With Tesla T4, hyperscale data centers can almost double their user density for Virtual Desktop Infrastructure (VDI) applications.

Table 2. Enhanced Video Engine, Tesla P4 versus Tesla T4

	Tesla P4 (70 W TDP)	Tesla T4 (70 W TDP)
H264 Decode (1080p30)	16 Streams	32 Streams
HEVC Decode (1080p30)	16 Streams	44 Streams
VP9 Decode (1080p30)	16 Streams	32 Streams

In addition to bringing revolutionary new features for high-end gaming and professional graphics, Turing also delivers new features such as multi-precision computing and significant increases in performance and energy efficiency for the data center. Along with other continual improvements in the NVIDIA Deep Learning Platform, such as the latest release of TensorRT 5.0 and CUDA 10, NVIDIA GPU-based inferencing solutions dramatically reduce the cost, size, and power consumption of data centers.

TURING MEMORY ARCHITECTURE AND DISPLAY FEATURES

This section dives deeper into key new memory hierarchy and display subsystem features of the Turing architecture.

Memory subsystem performance is crucial to application acceleration. Turing improves main memory, cache memory, and compression architectures to increase memory bandwidth and reduce access latency. Improved and enhanced GPU compute features help accelerate both games and many computationally intensive applications and algorithms. New display and video encode/decode features support higher resolution and HDR-capable displays, more advanced VR displays, increasing video streaming requirements in the datacenter, 8K video production, and other video-related applications. The following features are discussed in detail:

- GDDR6 Memory Subsystem
- L2 Cache and ROPs
- Turing Memory Compression
- Video and Display Engine
- USB-C and VirtualLink

GDDR6 Memory Subsystem

As display resolutions continue to increase and shader functionality and rendering techniques become more complex, memory bandwidth and size play a larger role in GPU performance. To maintain the highest possible frame rates and computational speed, the GPU not only needs more memory bandwidth, it also needs a large pool of memory to draw from to deliver sustained performance.

NVIDIA worked closely with the DRAM industry to develop the world’s first GPUs that use HBM2 and GDDR5X memories. Now Turing is the first GPU architecture to utilize GDDR6 memory.

GDDR6 is the next big advance in high-bandwidth GDDR DRAM memory design. Enhanced with many high-speed SerDes and RF techniques, GDDR6 memory interface circuits in Turing GPUs have been completely redesigned for speed, power efficiency, and noise reduction. This new interface design comes with many new circuit and signal training improvements that minimize noise and variations due to process, temperature, and supply voltage. Extensive clock gating was used to minimize power consumption during periods of lower utilization, resulting in significant overall power efficiency improvement. Turing's GDDR6 memory subsystem delivers 14 Gbps signaling rates and 20% power efficiency improvement over GDDR5X memory used in Pascal GPUs.

Achieving this speed increase requires end-to-end optimizations. Using extensive signal and power integrity simulations, NVIDIA carefully crafted Turing's package and board designs to meet the higher speed requirements. An example is a 40% reduction in signal crosstalk, which is one of the most severe impairments in large memory systems.

To realize speeds of 14 Gbps, every aspect of the memory subsystem was carefully crafted to meet the demanding standards that are required for such high frequency operation. Every signal in the design was carefully optimized to provide the cleanest memory interface signaling as possible (see Figure 11).

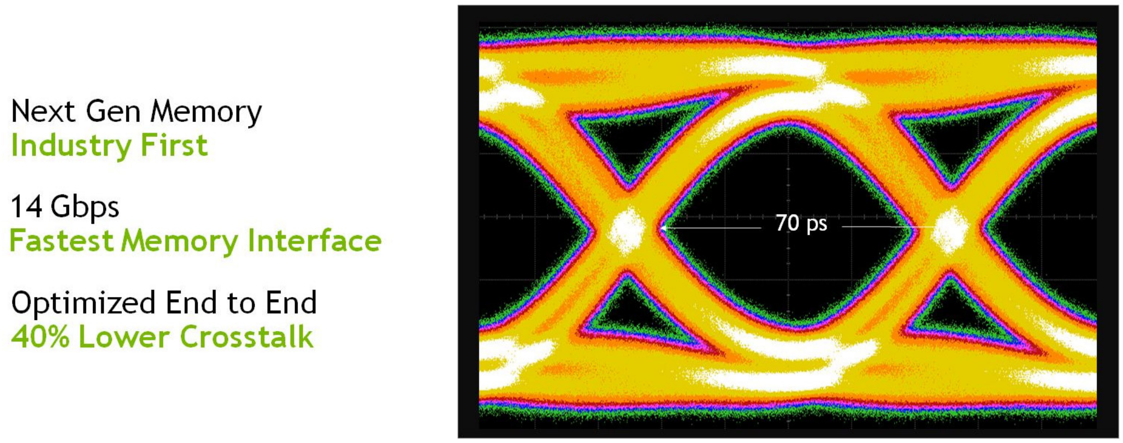


Figure 11. Turing GDDR6

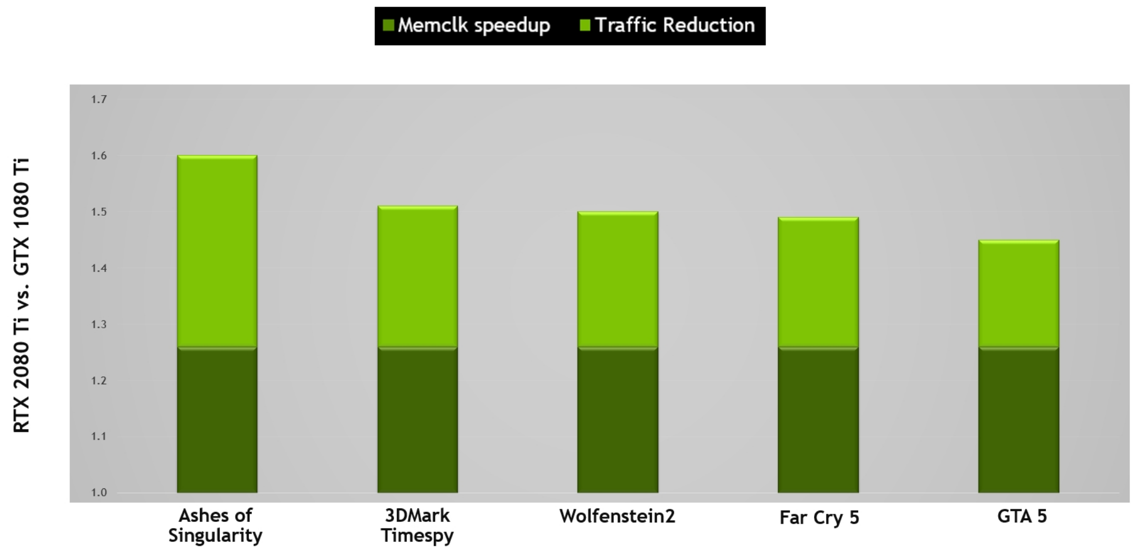
L2 Cache and ROPs

Turing GPUs add larger and faster L2 caches in addition to the new GDDR6 memory subsystem. The TU102 GPU ships with 6 MB of L2 cache, double the 3 MB of L2 cache that was offered in the prior generation GP102 GPU used in the TITAN Xp. TU102 also provides significantly higher L2 cache bandwidth than GP102.

Like prior generation NVIDIA GPUs, each ROP partition in Turing contains eight ROP units and each unit can process a single-color sample. A full TU102 chip contains 12 ROP partitions for a total of 96 ROPs.

Turing Memory Compression

NVIDIA GPUs utilize several lossless memory compression techniques to reduce memory bandwidth demands as data is written out to frame buffer memory. The GPU's compression engine has a variety of different algorithms which determine the most efficient way to compress the data based on its characteristics. This reduces the amount of data written out to memory and transferred from memory to the L2 cache and reduces the amount of data transferred between clients (such as the texture unit) and the frame buffer. Turing adds further improvements to Pascal's state-of-the-art memory compression algorithms, offering a further boost in effective bandwidth beyond the raw data transfer rate increases of GDDR6. As shown in Figure 12, the combination of raw bandwidth increases, and traffic reduction translates to a 50% increase in effective bandwidth on Turing compared to Pascal, which is critical to keep the architecture balanced and support the performance offered by the new Turing SM architecture.



The memory subsystem and compression (traffic reduction) improvements of Turing TU102-based RTX 2080 Ti deliver approximately 50% effective bandwidth improvements over the Pascal GP102-based 1080 Ti.

Figure 12. 50% Higher Effective Bandwidth

Video and Display Engine

Consumer demand for higher resolution displays continues to increase with every passing year. For example, 8K resolution (7680 x 4320) requires four times more pixels than 4K (3820 x 2160). Gamers and hardware enthusiasts also desire displays with higher refresh rates in addition to higher resolution to experience the smoothest possible image.

Turing GPUs include an all-new display engine designed for the new wave of displays, supporting higher resolutions, faster refresh rates, and HDR. Turing supports DisplayPort 1.4a allowing 8K resolution at 60 Hz and includes VESA's Display Stream Compression (DSC) 1.2 technology, providing higher compression that is visually lossless. Table 3 shows the DisplayPort support in the Turing GPUs.

Table 3. DisplayPort Support in Turing GPUs

	Bandwidth/Lane	Max Resolution Supported
DisplayPort 1.2	5.4 Gbps	4K @ 60 Hz
DisplayPort 1.3	8.1 Gbps	5K @ 60 Hz
DisplayPort 1.4a	8.1 Gbps	8K @ 60 Hz

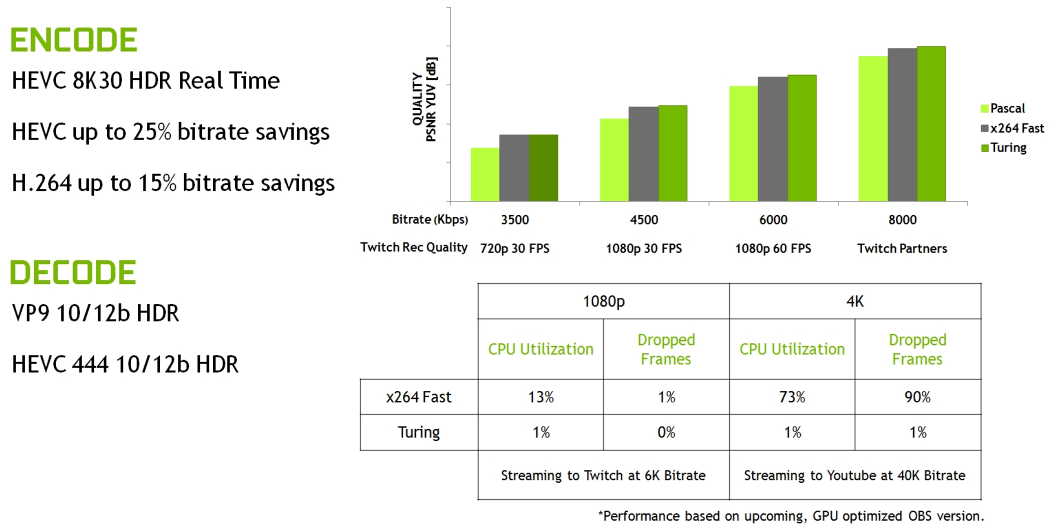
Turing GPUs can drive two 8K displays at 60 Hz with one cable for each display. 8K resolution can also be sent over USB-C (see *USB-C and VirtualLink* section on page 23 for more details).

Turing's new display engine supports HDR processing natively in the display pipeline. Tone mapping has also been added to the HDR pipeline. Tone mapping is a technique used to approximate the look of high dynamic range images on standard dynamic range displays. Turing supports the tone mapping formula defined by the *ITU-R Recommendation BT.2100* standard to avoid color shift on different HDR displays.

Turing GPUs also ship with an enhanced NVENC encoder unit that adds support for H.265 (HEVC) 8K encode at 30 fps. The new NVENC encoder provides up to 25% bitrate savings for HEVC and up to 15% bitrate savings for H.264.

Turing's new NVDEC decoder has been updated to support decoding of HEVC 4:4:4 8/10/12-bit video streams, and it also supports VP9 10/12-bit HDR similar to Pascal GP102/107/108 and Volta GV100 GPUs.

Turing improves encoding quality compared to prior generation Pascal GPUs and compared to software encoders. Figure 13 shows that on common Twitch and YouTube streaming settings, Turing's video encoder exceeds the quality of the software-based x264 encoder using the fast settings, with dramatically lower CPU utilization. 4K streaming is too heavy a workload for encoding on typical CPU setups, but Turing's encoder makes 4K streaming possible.



ENCODE

HEVC 8K30 HDR Real Time

HEVC up to 25% bitrate savings

H.264 up to 15% bitrate savings

DECODE

VP9 10/12b HDR

HEVC 444 10/12b HDR

	1080p		4K	
	CPU Utilization	Dropped Frames	CPU Utilization	Dropped Frames
x264 Fast	13%	1%	73%	90%
Turing	1%	0%	1%	1%
	Streaming to Twitch at 6K Bitrate		Streaming to Youtube at 40K Bitrate	

*Performance based on upcoming, GPU optimized OBS version.

New video features and video quality comparison of Turing to Pascal to a fast x264 software encoder

Figure 13. Video Feature Enhancements

USB-C AND VIRTUALLINK

Supporting VR headsets on today's PCs requires multiple cables to be connected between the headset and the system; a display cable to send image data from the GPU to the two displays in the headset, a cable to power the headset, and a USB connection to transfer camera streams and read back head pose information from the headset (to update frames rendered by the GPU). The number of cables can be uncomfortable for end users and limit their ability to move around when using the headset. Headset manufacturers need to accommodate the cables, complicating their designs and making them bulkier.

To address this issue, Turing GPUs are designed with hardware support for USB Type-C™ and VirtualLink™. VirtualLink is a new open industry standard that includes leading silicon, software, and headset manufacturers and is led by NVIDIA, Oculus, Valve, Microsoft, and AMD.

VirtualLink has been developed to meet the connectivity requirements of current and next-generation VR headsets. VirtualLink employs a new alternate mode of USB-C, designed to deliver the power, display, and data required to power VR headsets through a single USB-C connector.

VirtualLink simultaneously supports four lanes of High Bit Rate 3 (HBR3) DisplayPort along with the SuperSpeed USB 3 link to the headset for motion tracking. In comparison, USB-C only supports four lanes of HBR3 DisplayPort *OR* two lanes of HBR3 DisplayPort + two lanes SuperSpeed USB 3.

In addition to easing the setup hassles present in today's VR headsets, VirtualLink will bring VR to more devices. A single connector solution brings VR to small form factor devices that can accommodate a single, small footprint USB-C connector (such as a thin and light notebook) rather than today's VR infrastructure which requires a PC that can accommodate multiple connectors.

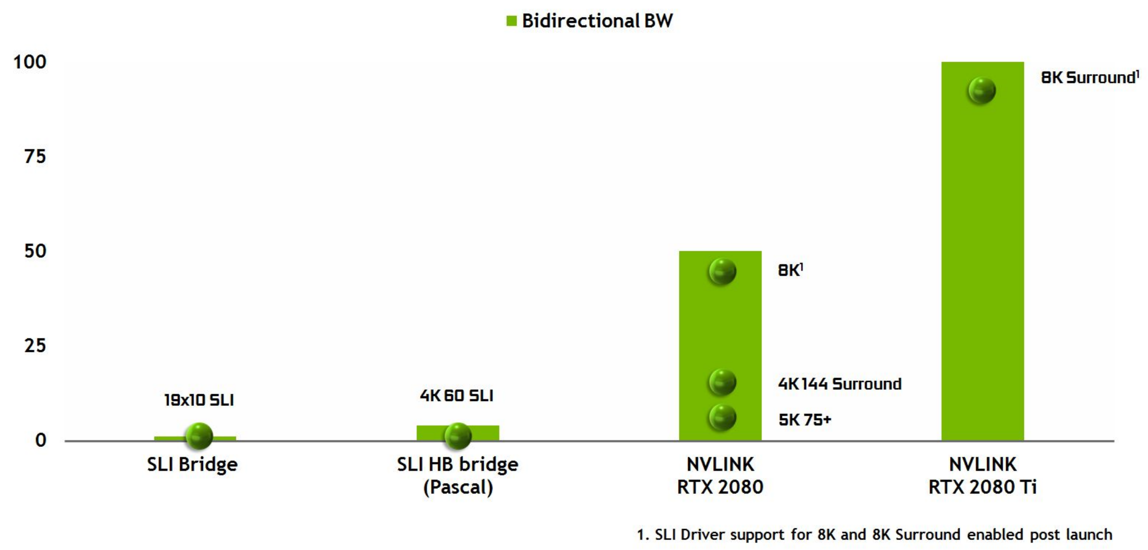
NVLINK IMPROVES SLI

Prior to the Pascal GPU architecture, NVIDIA GPUs used a single Multiple Input/Output (MIO) interface as the SLI Bridge technology to allow a second (or third or fourth) GPU to transfer its final rendered frame output to the primary GPU that was physically connected to a display. Pascal enhanced the SLI Bridge by using a faster dual-MIO interface, improving bandwidth between the GPUs, allowing higher resolution output, and multiple high-resolution monitors for NVIDIA Surround.

Note: Certain peer-to-peer inter-GPU SLI data transfers also occurred over the PCIe bus in Pascal and prior GPUs in some cases.

Turing TU102 and TU104 GPUs use NVLink instead of the MIO and PCIe interfaces for SLI GPU-to-GPU data transfers. The Turing TU102 GPU includes two x8 second-generation NVLink links, and Turing TU104 includes one x8 second-generation NVLink link. Each link provides 25 GB/sec peak bandwidth per direction between two GPUs (50 GB/sec bidirectional bandwidth). Two links in TU102 provides 50 GB/sec in each direction, or 100 GB/sec bidirectionally. Two-way SLI is supported with Turing GPUs that have NVLink, but 3-way and 4-way SLI configurations are not supported.

Compared to the previous SLI bridge, the increased bandwidth of the new NVLink bridge enables advanced display topologies that were not previously possible (see Figure 14).



Note: SLI driver support for 8K and 8K Surround will be enabled post-launch.

Figure 14. NVLink Enables New SLI Display Topologies

TURING RAY TRACING TECHNOLOGY

Ray tracing is a computationally-intensive rendering technology that realistically simulates the lighting of a scene and its objects. Turing GPU-based ray tracing technology can render physically correct reflections, refractions, shadows, and indirect lighting in real-time. See *Appendix D Ray Tracing Overview* on page 68 for a basic overview of how ray tracing works.

In the past, GPU architectures could not perform real time ray-tracing for games or graphical applications using a single GPU. While NVIDIA's GPU-accelerated NVIDIA Iray® plugins and OptiX ray tracing engine have delivered realistic ray-traced rendering to designers, artists, and technical directors for years, high quality ray tracing effects could not be performed in real-time. Similarly, current NVIDIA Volta GPUs can render realistic movie-quality ray-traced scenes, but not in real-time on a single GPU. Due to its processing intensive nature, ray tracing has not been used in games for any significant rendering tasks. Instead, games that require 30 to 90+ frame/second animations have relied on fast, GPU-accelerated rasterization rendering techniques for years, at the expense of fully realistic looking scenes.

Implementing real-time ray tracing on GPUs was an enormous technical challenge, requiring nearly 10 years of collaboration between NVIDIA's research, GPU hardware design, and software engineering teams. Real-time ray tracing in games and other applications is made possible by incorporation of multiple new hardware-based ray tracing acceleration engines called RT Cores in Turing TU102, TU104, and TU106 GPUs, combined with NVIDIA RTX software technology.

SOL MAN from NVIDIA SOL ray tracing demo running on a Turing TU102 GPU with NVIDIA RTX technology in real-time is shown in Figure 15 (see demo).

As mentioned, rasterization techniques have been the norm in real-time rendering for years, especially in computer games, and while many rasterized scenes can look very good, rasterization-based rendering has significant limitations. For example, rendering reflections and shadows using only rasterization requires simplifying assumptions that can cause many different types of artifacts. Similarly, static lightmaps may look correct until something moves, rasterized shadows often suffer from aliasing and light leaks, and screen-space reflections can only reflect off objects that are visible on the screen. These artifacts detract from the realism of the gaming experience and are costly for developers and artists to try to fix with additional effects.

Figure 15. SOL MAN from NVIDIA SOL Ray Tracing Demo (See Demo)

While ray tracing can produce much more realistic imagery than rasterization, it is also computationally intensive. We have found that the best approach is **hybrid rendering**, a combination of ray tracing and rasterization. With this approach, rasterization is used where it is most effective, and ray tracing is used where it provides the most visual benefit vs rasterization, such as rendering reflections, refractions, and shadows. Figure 16 Shows the hybrid rendering pipeline.

Hybrid Rendering combines ray tracing and rasterization techniques in the rendering pipeline to take advantage of what each does best to render a scene. SEED uses a hybrid rendering model for their PICA PICA real-time ray tracing experiment that features self-learning agents in a procedurally-assembled world. Built using SEED's R&D engine Halcyon, PICA PICA implements real-time ray tracing using Microsoft DXR and NVIDIA GPUs.

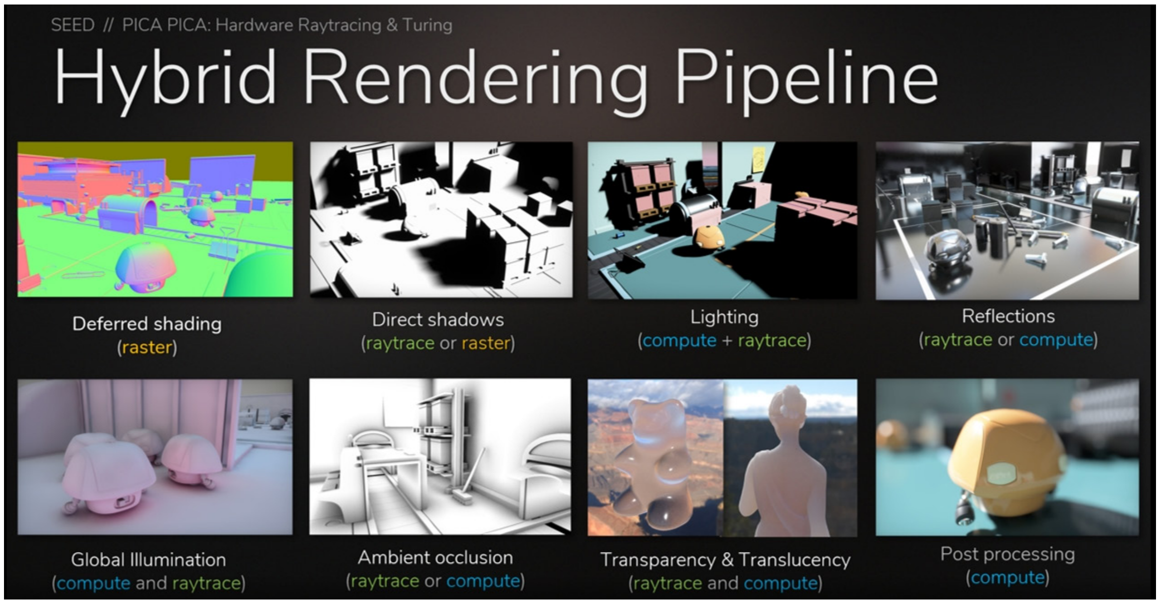

SEED // PICA PICA: Hardware Raytracing & Turing
Hybrid Rendering Pipeline
Deferred shading (raster)
Direct shadows (raytrace or raster)
Lighting (compute + raytrace)
Reflections (raytrace or compute)
Global Illumination (compute and raytrace)
Ambient occlusion (raytrace or compute)
Transparency & Translucency (raytrace and compute)
Post processing (compute)

Image Courtesy of the SEED division of EA (SEED//Pica Pica Hardware Raytracing and Turing)

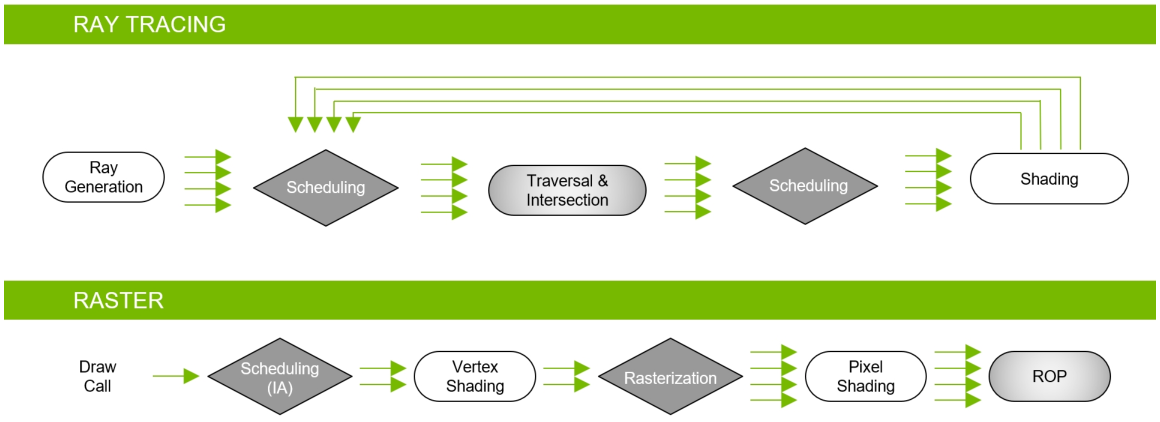
Figure 16. Hybrid Rendering Pipeline

Rasterization and z-buffering is much faster at determining object visibility and can substitute for the primary ray casting stage of the ray tracing process. Ray tracing can then be used for shooting secondary rays to generate high-quality physically correct reflections, refractions, and shadows.

Developers can also use material property thresholds to determine areas to perform ray tracing in a scene. One technique might be to specify that only surfaces with a certain reflectivity level, say 70%, would trigger whether ray tracing should be used on that surface to generate secondary rays.

We expect many developers to use hybrid rasterization/ray tracing techniques to attain high frame rates with excellent image quality. Alternatively, for professional applications where image fidelity is the highest priority, we expect to see use of ray tracing for the entire rendering workload, casting primary and secondary rays to create amazingly realistic rendering.

Turing GPUs not only include dedicated ray tracing acceleration hardware, but also use an advanced acceleration structure described in the next section. Essentially, an entirely new rendering pipeline is available to enable real-time ray tracing in games and other graphics applications using a single Turing GPU (see Figure 17).



Both Ray tracing and Rasterization pipeline operate simultaneously and cooperatively in Hybrid Rendering model used in Turing GPUs.

Figure 17. Details of Ray Tracing and Rasterization Pipeline Stages

While Turing GPUs enable real time ray tracing, the number of primary or secondary rays cast per pixel or surface location varies based on many factors, including scene complexity, resolution, other graphics effects rendered in a scene, and of course GPU horsepower. Do not expect hundreds of rays cast per pixel in real-time. In fact, far fewer rays are needed per pixel when using Turing RT Core acceleration in combination with advanced denoising filtering techniques. NVIDIA Real-Time Ray Tracing Denoiser modules can significantly reduce the number of rays required per pixel and still produce excellent results.

Real-time ray tracing of selected objects can make many scenes in games and applications look as realistic as high-end movie special effects, or as good as ray-traced images created with professional software-based non-real-time rendering applications. Figure 18 shows an example from the Reflections demo created by Epic Games in collaboration with ILMxLAB and NVIDIA. Ray-traced reflections, ray-traced area light shadows, and ray-traced ambient occlusion can run on a single Quadro RTX 6000 or GeForce RTX 2080 Ti GPU delivering rendering quality nearly indistinguishable from movies.

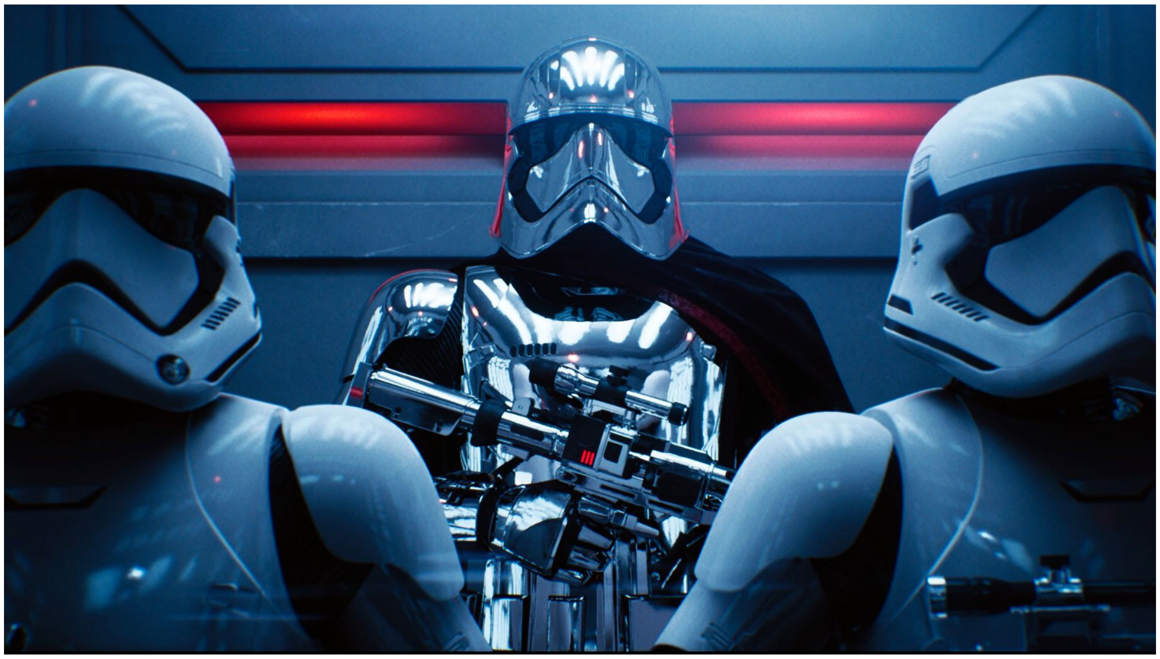


Figure 18. From Reflections Demo

Turing ray tracing hardware works with NVIDIA's RTX ray tracing technology, NVIDIA Real-Time Ray Tracing Libraries, NVIDIA OptiX, the Microsoft DXR API, and the soon-to-come Vulkan ray tracing API. Users will experience real-time, cinematic-quality ray-traced objects and characters in games at playable frame-rates, or visual realism in professional graphics applications that has been impossible with prior GPU architectures in real time.

Turing GPUs can accelerate ray tracing techniques used in many of the following rendering and non-rendering operations:

- Reflections and Refractions
- Shadows and Ambient Occlusion
- Global Illumination
- Instant and off-line lightmap baking
- Beauty shots and high-quality previews
- Primary rays for foveated VR rendering
- Occlusion Culling
- Physics, Collision Detection, Particle simulations
- Audio simulation (ex., NVIDIA VRWorks Audio built on top of the OptiX API)
- AI visibility queries
- In-engine Path Tracing (non-real-time) to generate reference screenshots for tuning real-time rendering techniques and denoisers, material composition, and scene lighting.

More detail is presented on rendering ray-traced shadows, ambient occlusion, and reflections using Turing ray tracing acceleration in following sections. The NVIDIA Developer Site has more details describing rendering operations that can be accelerated with Turing ray tracing.

TURING RT CORES

At the heart of Turing's hardware-based ray tracing acceleration is the new RT Core included in each SM. RT Cores accelerate Bounding Volume Hierarchy (BVH) traversal and ray/triangle intersection testing (ray casting) functions. (See *Appendix D Ray Tracing Overview* on page 68 for more details on how BVH acceleration structures work). RT Cores perform visibility testing on behalf of threads running in the SM.

RT Cores work together with advanced denoising filtering, a highly-efficient BVH acceleration structure developed by NVIDIA Research, and RTX compatible APIs to achieve real time ray tracing on single Turing GPU. RT Cores traverse the BVH autonomously, and by accelerating traversal and ray/triangle intersection tests, they offload the SM, allowing it to handle other vertex, pixel, and compute shading work. Functions such as BVH building and refitting are handled by the driver, and ray generation and shading is managed by the application through new types of shaders.

To better understand the function of RT Cores, and what exactly they accelerate, we should first explain how ray tracing is performed on GPUs or CPUs without a dedicated hardware ray tracing engine. Essentially, the process of BVH traversal would need to be performed by shader operations and take thousands of instruction slots per ray cast to test against bounding box intersections in the BVH until finally hitting a triangle and the color at the point of intersection contributes to final pixel color (or if no triangle is hit, background color may be used to shade a pixel).

Ray tracing without hardware acceleration requires thousands of software instruction slots per ray to test successively smaller bounding boxes in the BVH structure until possibly hitting a triangle. It's a computationally intensive process making it impossible to do on GPUs in real-time without hardware-based ray tracing acceleration (see Figure 19).

The RT Cores in Turing can process all the BVH traversal and ray-triangle intersection testing, saving the SM from spending the thousands of instruction slots per ray, which could be an enormous amount of instructions for an entire scene. The RT Core includes two specialized units. The first unit does bounding box tests, and the second unit does ray-triangle intersection tests. The SM only has to launch a ray probe, and the RT core does the BVH traversal and ray-triangle tests, and return a hit or no hit to the SM. The SM is largely freed up to do other graphics or compute work. See Figure 20 or an illustration of Turing ray tracing with RT Cores.

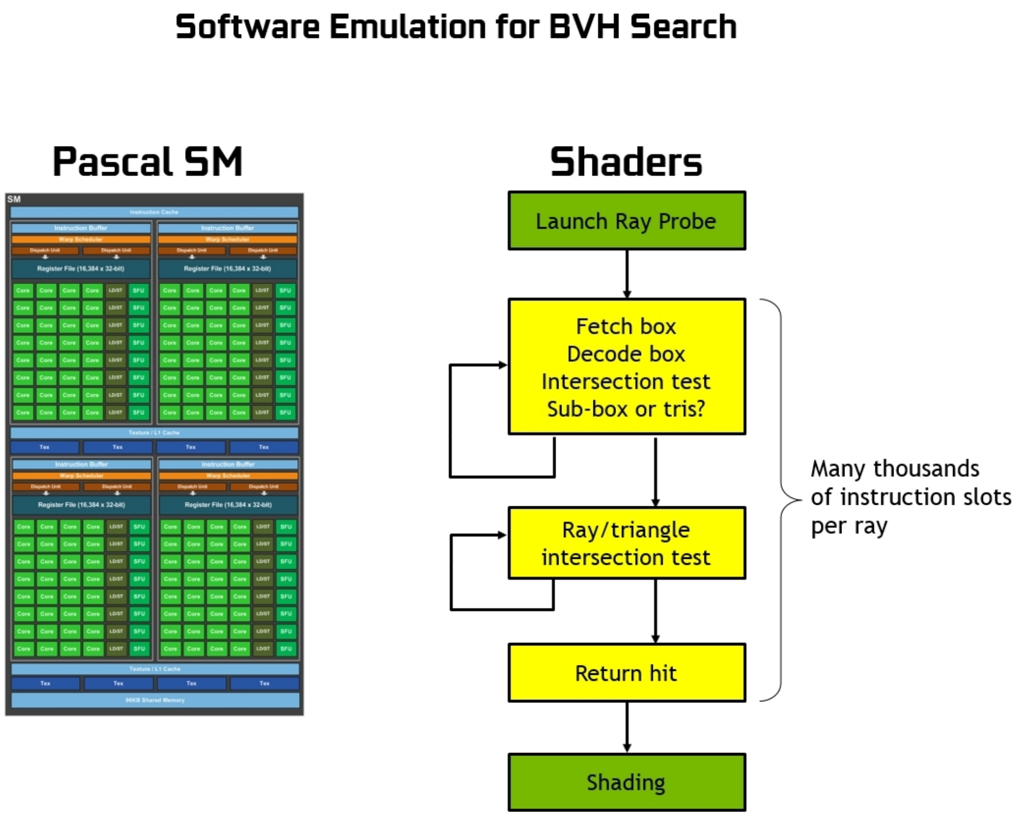


Figure 19. Ray Tracing Pre Turing

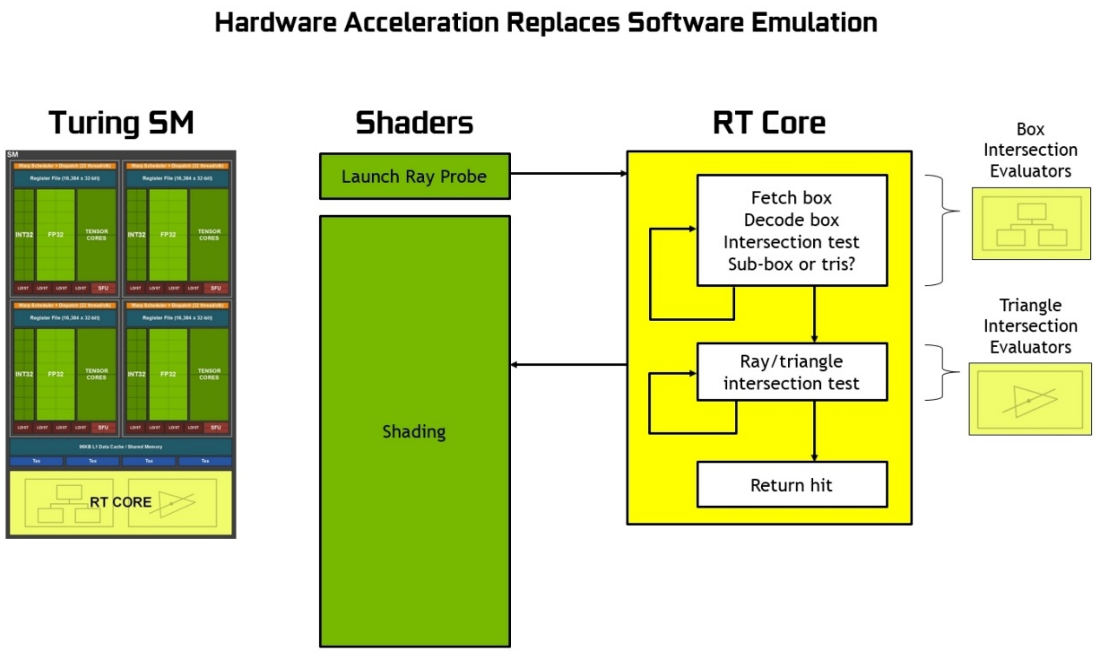
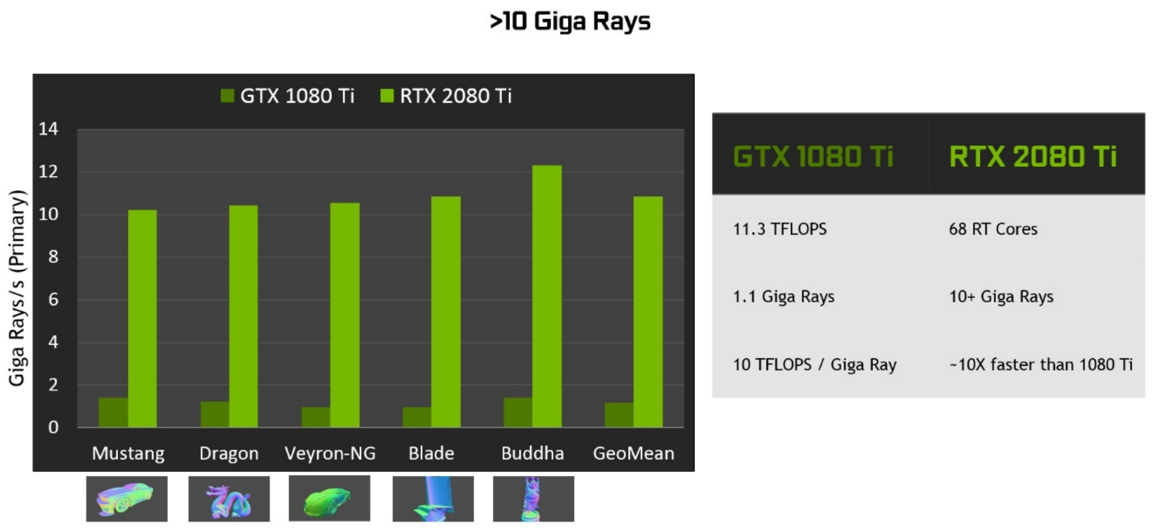


Figure 20. Turing Ray Tracing with RT Cores

Turing ray tracing performance with RT Cores is significantly faster than ray tracing in Pascal GPUs. Turing can deliver far more Giga Rays/Sec than Pascal on different workloads, as shown in Figure 21. Pascal is spending approximately 1.1 Giga Rays/Sec, or 10 TFLOPS / Giga Ray to do ray tracing in software, whereas Turing can do 10+ Giga Rays/Sec using RT Cores, and run ray tracing 10 times faster.

Note: This paper does not cover developer details for implementing ray tracing in games or applications with RTX, DXR, or other APIs, but many resources exist with such information. Good initial information sources include Introduction to NVIDIA RTX and DirectX Ray Tracing blog post, the NVIDIA RTX Technology developer site, and a publicly accessible GDC 2018 course on RTX presented by NVIDIA called *Ray Tracing in Games with NVIDIA RTX*. Also refer to Microsoft's blog on DXR.



GTX 1080 Ti	RTX 2080 Ti
11.3 TFLOPS	68 RT Cores
1.1 Giga Rays	10+ Giga Rays
10 TFLOPS / Giga Ray	~10X faster than 1080 Ti

Figure 21. Turing Ray Tracing Performance

NVIDIA NGX TECHNOLOGY

NVIDIA NGX™ (Neural Graphics Acceleration) is the new deep learning-based technology stack which is part of NVIDIA RTX Technology. NGX utilizes deep neural networks (DNNs) and a set of *Neural Services* to perform AI-based functions that accelerate and enhance graphics, rendering, and other client-side applications. NGX employs the Turing Tensor Cores for deep learning-based operations and accelerates delivery of NVIDIA deep learning research directly to the end-user. Note that NGX does not work on GPU architectures before Turing.

Note: NGX does not function on GPU architectures prior to Turing.

NGX SOFTWARE ARCHITECTURE

The features of NGX tightly couple to the NVIDIA driver and hardware. The NGX API provides access to several AI features for games and applications. The features are pre-trained by NVIDIA and ready for integration. The API has been designed to be thin and easy for applications to integrate multiple AI features. NGX services run on the GPU, allowing it to support multiple features and applications.

NVIDIA NGX features are managed by the NVIDIA GeForce Experience™ (GFE) application or the tech preview version of the NVIDIA Quadro Experience™ (QXP) application. After GFE or QXP is installed or updated, it looks for the presence of a Turing GPU. Once detected, the *NGX Core* package is downloaded and installed. GFE/QXP communicates with NGX Core to determine the game and application IDs present and their relevance to NGX. Different DNN models that work with various installed games and applications are then downloaded for subsequent use.

NGX DNN models can interface with CUDA 10, the DirectX and Vulkan drivers, as well as take advantage of NVIDIA TensorRT™, the high-performance deep learning inference optimizer that delivers low latency and high-throughput for deep learning inference applications. NGX models and services are accelerated by Turing's enhanced Tensor Cores.

DEEP LEARNING SUPER-SAMPLING (DLSS)

In modern games, rendered frames are not displayed directly, rather they go through a post processing image enhancement step that combines input from multiple rendered frames, trying to remove visual artifacts such as aliasing while preserving detail. For example, Temporal Anti-Aliasing (TAA), a shader-based algorithm that combines two frames using motion vectors to determine where to sample the previous frame, is one of the most common image enhancement algorithms in use today. However, this image enhancement process is fundamentally very difficult to get right.

NVIDIA's researchers recognized that this type of problem - an image analysis and optimization problem with no clean algorithmic solution - would be a perfect application for AI. As discussed earlier in this document, image processing cases (for example ImageNet) are among the biggest successful applications of deep learning. Deep learning has now achieved super-human ability to recognize dogs, cats, birds etc., from looking at the raw pixels in an image. In this case, the goal would be to combine rendered images, based on looking at raw pixels, to produce a high-quality result—a different objective but using similar capabilities.

The deep neural network (DNN) that was developed to solve this challenge is called Deep Learning Super-Sampling (DLSS). DLSS produces a much higher quality output than TAA from a given set of input samples, and we leverage this capability to improve overall performance. Whereas TAA renders at the final target resolution and then combines frames, subtracting detail, DLSS allows faster rendering at a lower input sample count, and then infers a result that at target resolution is similar quality to the TAA result, but with roughly half the shading work.

Figure 22, shows a sampling of results on the UE4 Infiltrator demo. DLSS provides image quality that is similar to TAA, with much improved performance. The much faster raw rendering horsepower of RTX 2080 Ti, combined with the performance uplift from DLSS and Tensor Cores, enables RTX 2080 Ti to achieve 2x the performance of GTX 1080 Ti.

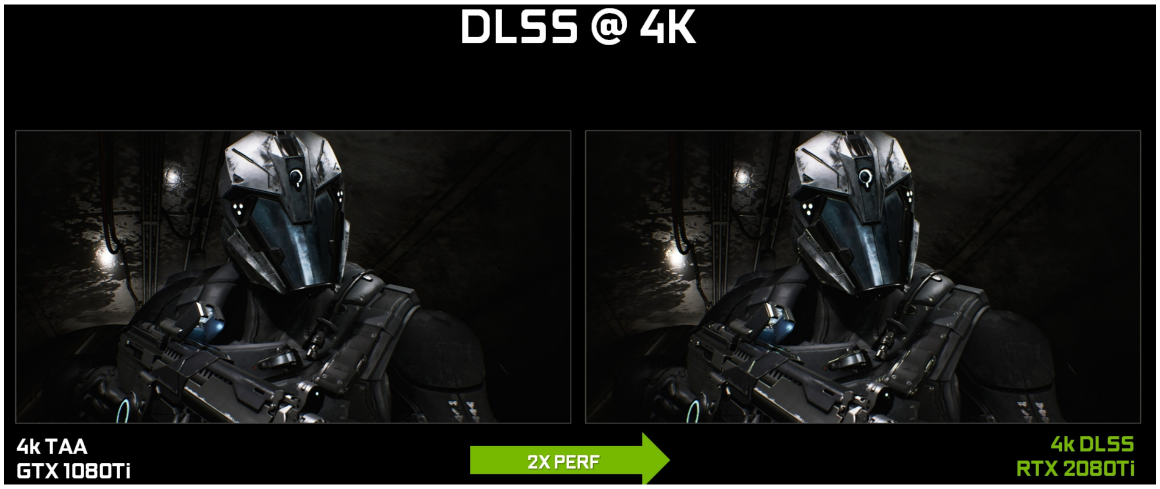


Figure 22. Turing with 4K DLSS is Twice the Performance of Pascal with 4K TAA

The key to this result is the training process for DLSS, where it gets the opportunity to learn how to produce the desired output based on large numbers of super-high-quality examples. To train the network, we collect thousands of “ground truth” reference images rendered with the gold standard method for perfect image quality, 64x supersampling (64xSS). 64x supersampling means that instead of shading each pixel once, we shade at 64 different offsets within the pixel, and then combine the outputs, producing a resulting image with ideal detail and anti-aliasing quality. We also capture matching raw input images rendered normally. Next, we start training the DLSS network to match the 64xSS output frames, by going through each input, asking DLSS to produce an output, measuring the difference between its output and the 64xSS target, and adjusting the weights in the network based on the differences, through a process called *back propagation*. After many iterations, DLSS learns on its own to produce results that closely approximate the quality of 64xSS, while also learning to avoid the problems with blurring, disocclusion, and transparency that affect classical approaches like TAA.

In addition to the DLSS capability described above, which is the standard DLSS mode, we provide a second mode, called DLSS 2X. In this case, DLSS input is rendered at the final target resolution and then combined by a larger DLSS network to produce an output image that approaches the level of the 64x super sample rendering - a result that would be impossible to achieve in real time by any traditional means. Figure 23 shows DLSS 2X mode in operation, providing image quality very close to the reference 64x super-sampled image.

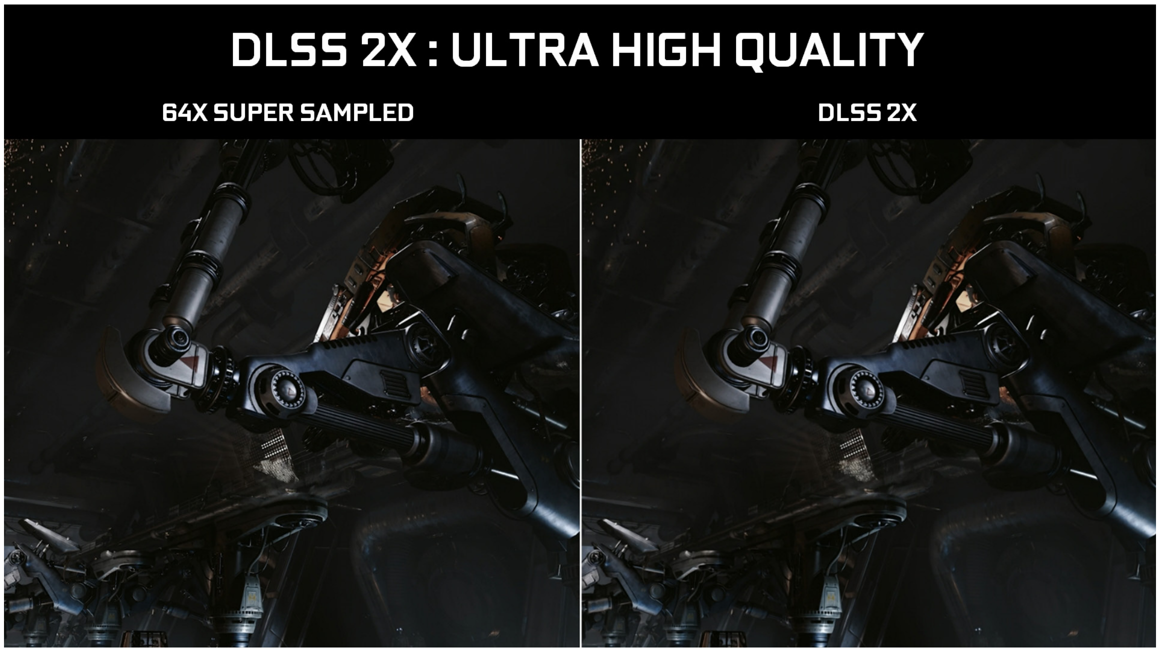


Figure 23. DLSS 2X versus 64xSS image almost Indistinguishable

Finally, Figure 24 illustrates one of the challenging cases for multi frame image enhancement. In this case, a semi-transparent screen floats in front of a background that is moving differently. TAA tends to blindly follow the motion vectors of the moving object, blurring the detail on the screen. DLSS is able to recognize that changes in the scene are more complex and combines the inputs in a more intelligent way that avoids the blurring issue.

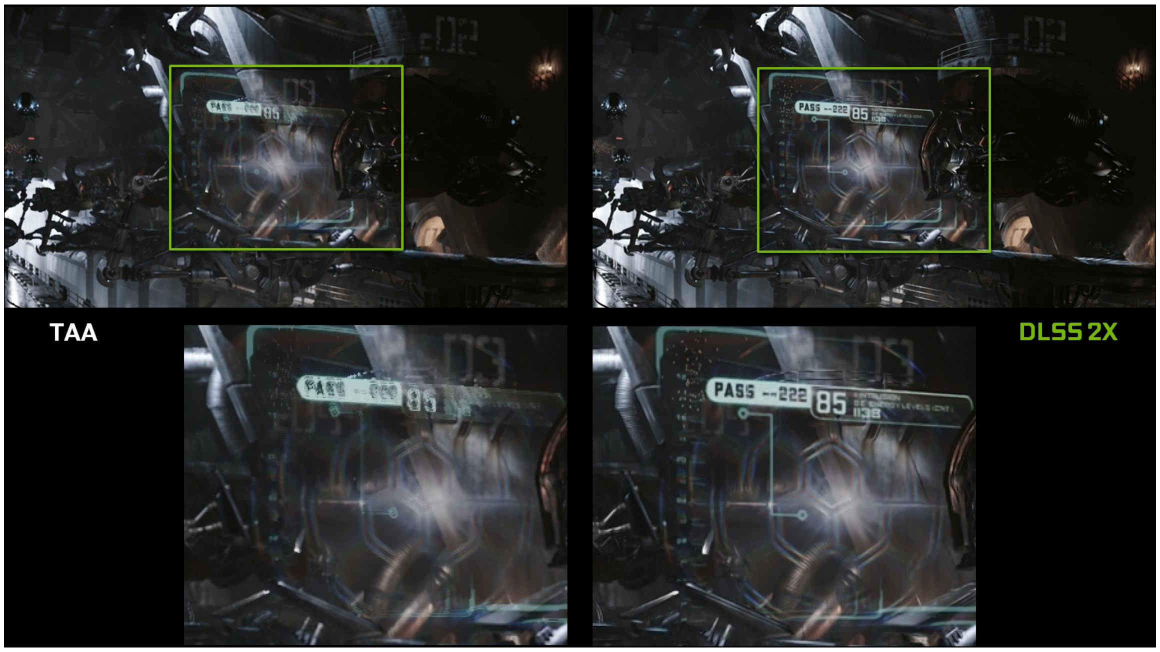


Figure 24. DLSS 2X Provides Significantly Better Temporal Stability and Image Clarity Than TAA

INPAINTING

InPainting allows an application to provide features to remove existing content from an image and use an NGX AI algorithm to replace the removed content with a realistic computer-generated alternative. For example, Inpainting could be used to automatically remove power lines from a landscape image, replacing them seamlessly with the existing sky background. The concept of inpainting is not new, but existing solutions rely on copying data from somewhere within the image to fill the hole. This can lead to visually obvious tiling patterns if the algorithm is not tuned well. The NGX InPainting algorithm instead relies on the training from a large set of real-world images to synthesize new content to fill the gap. The result is a more visually meaningful picture (see Figure 25).

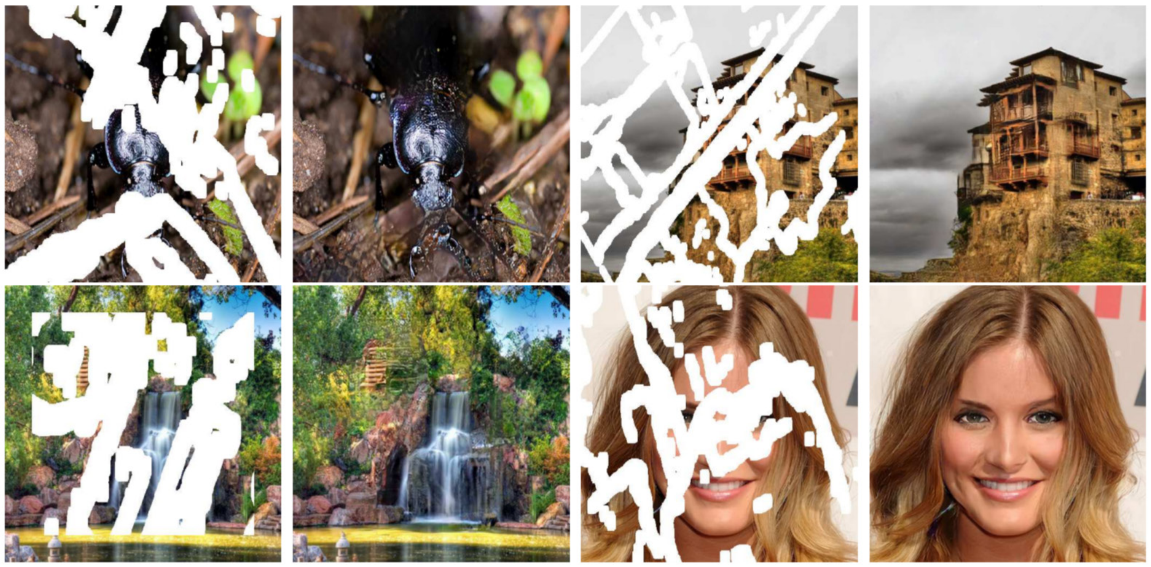


Figure 25. NGX InPainting Examples, Missing Image Data Is Intelligently Replaced with Meaningful Image Information

AI SLOW-MO

AI Slow-Mo inserts interpolated frames into a video stream to provide smooth, slow-motion video. NGX analyzes frames for features and objects, identifies object and camera movement, and creates new video frames between the existing video frames. The result in smooth slow-motion video with reduced interpolation artifacts. Watch this NVIDIA Research video to see AI Slow-Mo in action.

AI SUPER REZ

AI Super Rez increases the resolution of an image or video by 2x, 4x or 8x. Unlike traditional filtering methods which stretch out the existing pixels and filter between them, AI Super Rez creates new pixels by interpreting the image and intelligently placing data (see Figure 26). This results in a sharper enlargement that correctly preserves the depth of field and other artistic aspects. The video super-resolution network, which is highly optimized, can run in real-time (~30 fps) for 1080p to 4K upscaling, with PSNR 1-2 dB higher than bicubic interpolation.

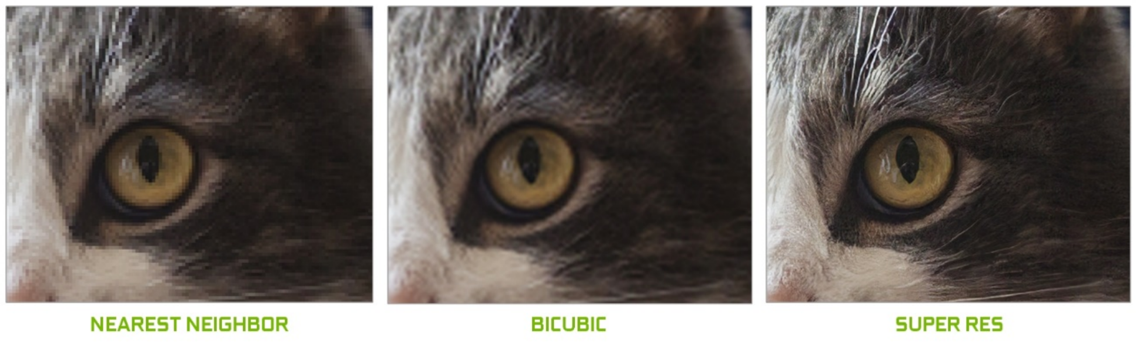


Figure 26. AI Super Rez Provides Improved Image Clarity Over Other Filtering Methods

TURING ADVANCED SHADING TECHNOLOGIES

MESH SHADING

The real world is a visually rich, geometrically complex place. Outdoor scenes in particular can be composed of hundreds of thousands of elements (rocks, trees etc.). CAD models present similar challenges. Today’s graphics pipeline with vertex, tessellation, and geometry shaders is very effective at rendering the details of a single object, but still has limitations. Each object requires its own unique draw call from the CPU and the shader model is a per-thread model which limits the types of algorithms that can be used. Mesh Shading introduces a new, more flexible model that enables developers to eliminate CPU draw call bottlenecks and use more efficient algorithms for producing triangles.

Visually rich images, like those shown in Figure 27, have too many unique complex objects to render in real time with today’s graphics pipeline.

Figure 27. Mesh Shading, Visually Rich Images

Figure 28 shows the Mesh Shading based pipeline compared to today's full geometry processing pipeline. Today, developers can either use vertex shaders to directly produce triangles for the rasterizer, or they can use tessellation shaders to process patches which get tessellated to produce final triangles for rasterization.

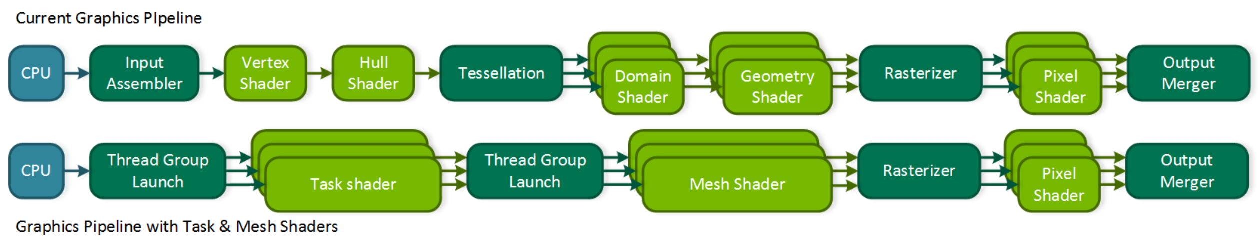
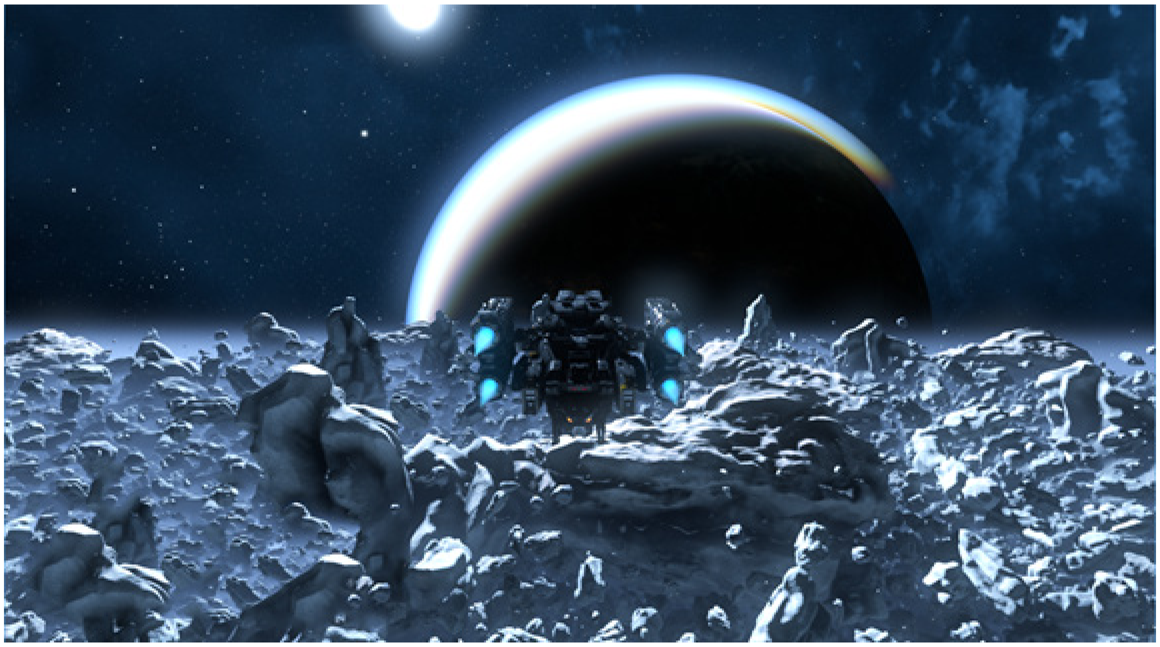


Figure 28. Current Graphics Pipeline versus a Graphics Pipeline with Task and Mesh Shaders

Mesh Shading introduces two new shader stages, Task Shaders and Mesh Shaders, that support this same functionality, but with much more flexibility. The mesh shader stage produces triangles for the rasterizer, but internally, instead of using a single-thread program model, it uses a cooperative thread model similar to compute shaders. Ahead of the mesh shader in the pipeline is the task shader. The task shader operates similarly to the hull shader stage of tessellation, in that it is able to dynamically generate work. However, like the mesh shader, it uses a cooperative thread model and instead of having to take a patch as input and tessellation decisions as output, its input and output are user defined.

Figure 29 shows one example of the power of mesh shading. This scene is rendering a challenging environment where the viewer's perspective is taking place in a wide field of view with hundreds of thousands of individual objects. Instead of sending each object to the GPU with a unique draw call from the CPU, a developer can now send the GPU a list of many objects. The Task Shader then process this object list in parallel and launches Mesh Shaders to shade corresponding triangles and submit them to the rasterizer. This approach eliminates the CPU bottleneck for object processing and enables an increase of more than an order of magnitude in the number of objects that can be displayed at real time frame rates.

Figure 30 shows another optimization that is supported by mesh shading. In Figure 29 we want each asteroid to have realistic detail when viewed up close, but many of the asteroids are too far away from the viewer for any detail to be visible. One approach for optimizing this case is to have multiple versions of each object available (at different levels of detail) and to pick the appropriate version on the fly based on knowing the size of the object in screen space in the current frame. The Task Shader supports this optimization. As it scans an object list it can also look at the size of each object and pick the appropriate LOD version, sending it down to the Mesh Shader for processing, or in the case of tessellation, it can tell the Mesh Shader to further tessellate triangles from that LOD version.



Demonstrates the use of mesh shading to render hundreds of thousands of objects in real time

Figure 29. Screenshot from the Asteroid Field Demo

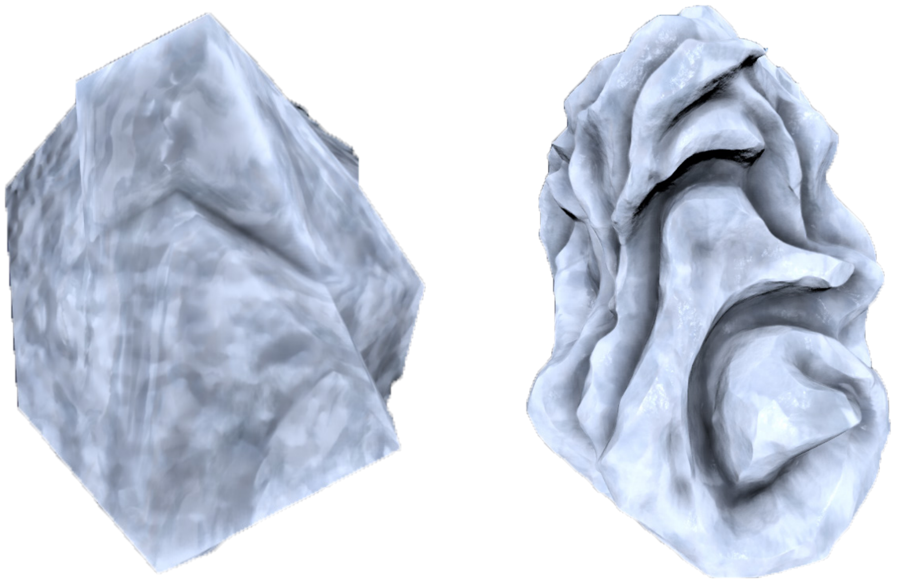


Figure 30. An Asteroid at Low and High Levels of Detail (LOD)

Mesh shading also makes it easier for developers to manipulate geometry in unusual ways. Figure 31 shows mesh shading used to implement a dynamic cutaway function for CAD applications. Any geometry that is inside the spherical cutaway region defined by the spherical boundary is eliminated, revealing the detailed structure of elements of the car underneath that region. The mesh and task shaders can perform this operation by culling and modifying geometry based on its position relative to the sphere.

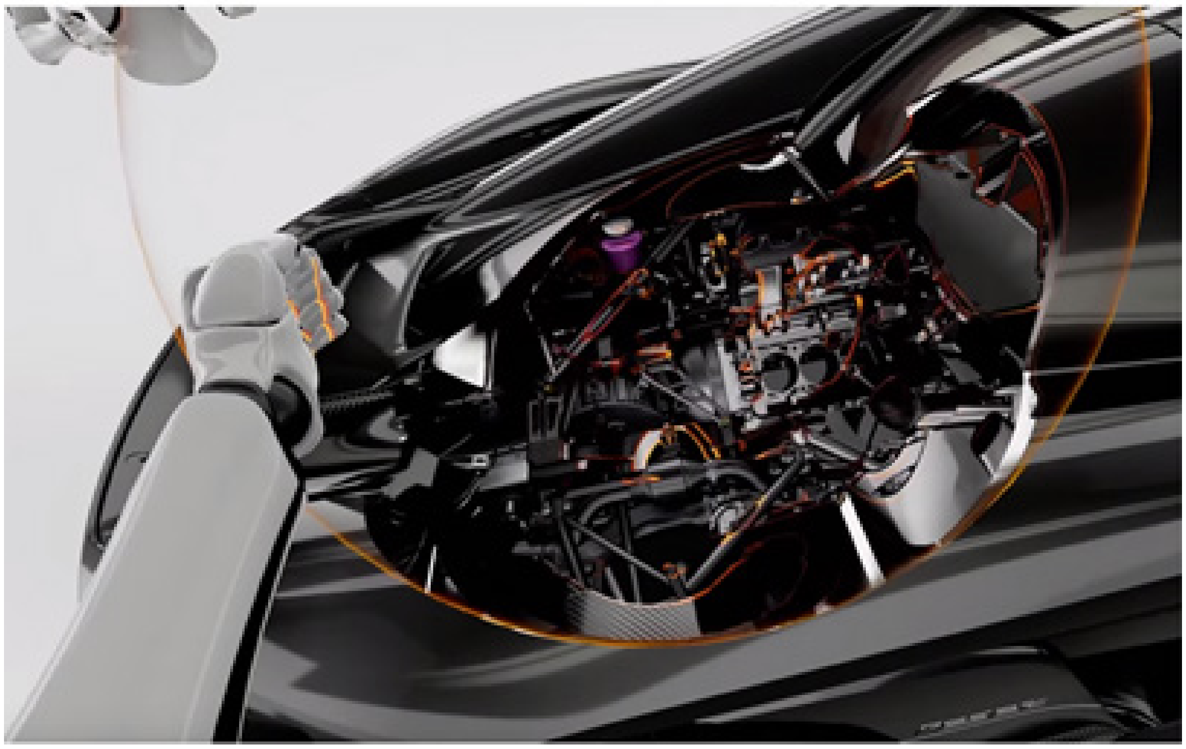


Figure 31. Dynamically Computed, Spherical Cutaway of a Koenigsegg Model, Viewed in NVIDIA Holodeck™

VARIABLE RATE SHADING

As the overall demand for compute horsepower continues to increase with each new generation of games, we are always looking for methods that would allow developers to eliminate shading work that does not improve the image quality of the final rendered frame. Full GPU horsepower should only be applied to improving the user experience, either with richer graphics or higher framerates.

In previous generations, we have introduced techniques including Multi-Resolution Shading (MRS) and Lens-Matched Shading (LMS) to optimize shading workloads particularly related to VR. An important property of VR systems is that the optics in the lens system has a varied resolution and sample rate of the view surface. MRS and LMS allowed developers to split the rendering surface up into 16 subregions and match the sampling rate to the lens in each region, rather than over shading everywhere to meet the maximum local sampling needs.

However, this is only one example of a general problem. Turing introduces a new and dramatically more flexible capability for controlling shading rate called *Variable Rate Shading* (VRS). With VRS, shading rate can now be adjusted dynamically at an extremely fine level—every 16-pixel x 16-pixel region of the screen can now have a different shading rate (see Figure 32).

This fine-level of control enables developers to deploy new algorithms that were not previously possible for optimizing shading rate and increasing performance. This section discusses the underlying hardware mechanisms of VRS, and a few of the powerful new algorithms that it enables.

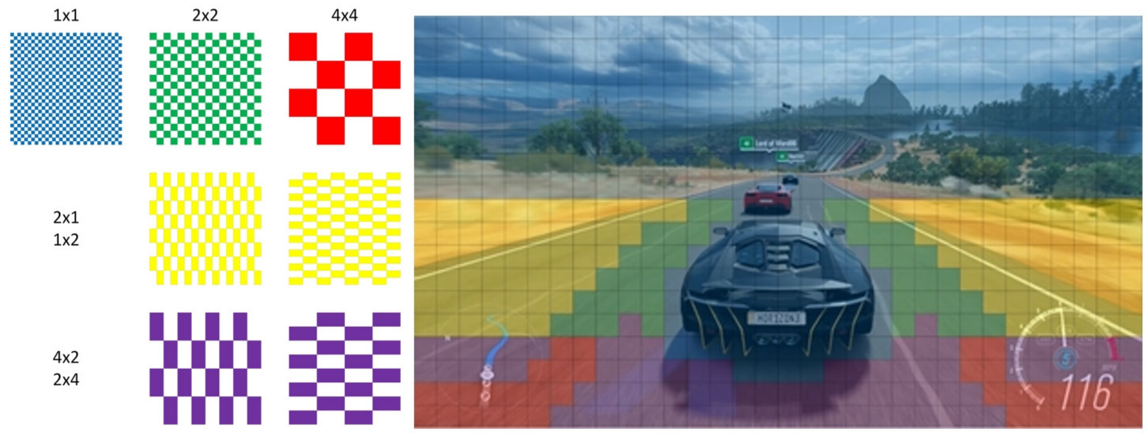


Figure 32. Turing VRS Supported Shading Rates and Example Application to a Game Frame

Without VRS, every pixel in the scene in Figure 32 would be shaded individually (the 1 x 1 blue grid case). With VRS, the pixel shading rate of triangles can vary. The developer has up to seven options to choose from for each 16x16 pixel region, including having one shading result be used to color four pixels (2 x 2), or 16 pixels (4 x 4), or non-square footprints like 1 x 2 or 2 x 4. The colored overlay on the right side of Figure 32 shows a possible application to a frame—perhaps the car could be shaded at full rate (blue region) while the area near the car could be shaded once per four pixels (green), and the road to the left and right could be shaded once per eight pixels (yellow).

Overall, with Turing's VRS technology, a scene can be shaded with a mixture of rates varying between once per visibility sample (super-sampling) and once per sixteen visibility samples. The developer can specify shading rate spatially (using a texture) and using a per-primitive shading rate attribute. As a result, a single triangle can be shaded using multiple rates, providing the developer with fine-grained control.

Note: VRS allows the developer to control the shading rate without changing the visibility rate. The ability to decouple shading rate and visibility rate makes VRS more broadly applicable than techniques such as MRS and LMS, that lower total rendering resolution in specified regions. At the same time, VRS and MRS/LMS can be used in combination since they are independent techniques enabled by separate hardware paths.

The following three powerful algorithms leverage VRS:

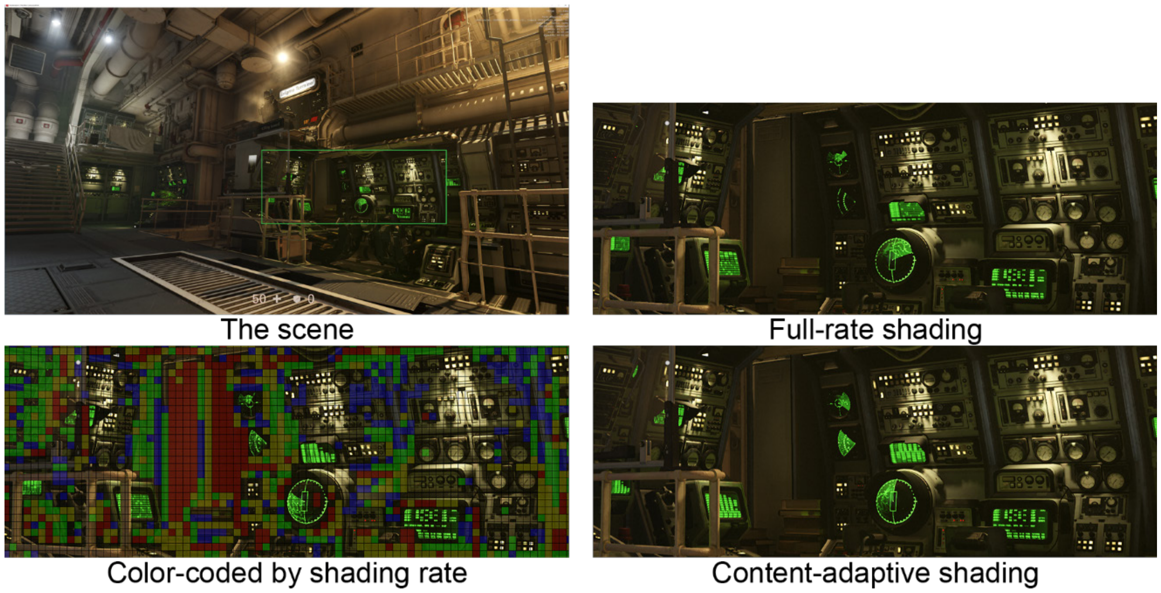
- Content Adaptive Shading
  Reduces shading rate in regions of slowly changing color.
- Motion Adaptive Shading
  Variably decreases shading rate of moving objects.
- Foveated Rendering
  Reduces shading rate in areas away from the viewer’s focus.

Content Adaptive Shading

In Content Adaptive Shading, shading rate is simply lowered by considering factors like spatial and temporal (across frames) color coherence. The desired shading rate for different parts of the next frame to be rendered are computed in a post-processing step at the end of the current frame. If the amount of detail in a particular region was relatively low (sky or a flat wall etc.), then the shading rate can be locally lowered in the next frame. The output of the post-process analysis is a texture specifying a shading rate per 16 x 16 tile, and this texture is used to drive shading rate in the next frame. A developer can implement content-based shading rate reduction without modifying their existing pipeline, and with only small changes to their shaders.

Figure 31 shows an example application of Content Adaptive Shading. Of course, watching Content Adaptive Shading in action in real time is the best way to appreciate its effectiveness, but for illustrating its operation some screen shots were used. The green box in the upper left full screen image shows a crop area that is zoomed into. The lower left shows this area magnified, with a shading rate overlay as described in Figure 32. Note that the flat vertical wall is shaded at the lowest rate (red = 4 x 4), while the gauges and dials are shaded at full rate (no color overlay = 1 x 1), and various intermediate rates are used elsewhere in the scene.

On the right side, the upper and lower images show screenshots of this cropped area with Content Adaptive Shading off (top) versus on (bottom), with no visual difference in image quality (the images are slightly different due to different sampling times versus the instrument panel animations).



Content Adaptive Shading in the lower right image is created by applying different shading rates (lower left image) to the original scene data. Note the similarity between the scene shaded at full shading rate compared to the scene shaded with Content Adaptive Shading

Figure 33. Example of Content Adaptive Shading

Motion Adaptive Shading

The second application of Variable Rate Shading exploits object motion. Our eyes are designed to track moving objects linearly, so that we can see their details even when in motion. However, objects on LCD screens do not move smoothly or continuously. Rather, they jump from one location to the next with each 60 Hz frame update. From the perspective of our eye, which is trying to smoothly track the object, it looks like it is wiggling back and forth on the retina as its location moves ahead and behind of the path the eye is tracking. The net result is that we cannot see the full detail of the object, instead we see a somewhat lower resolution/blurred version. Figure 34 illustrates this scenario.

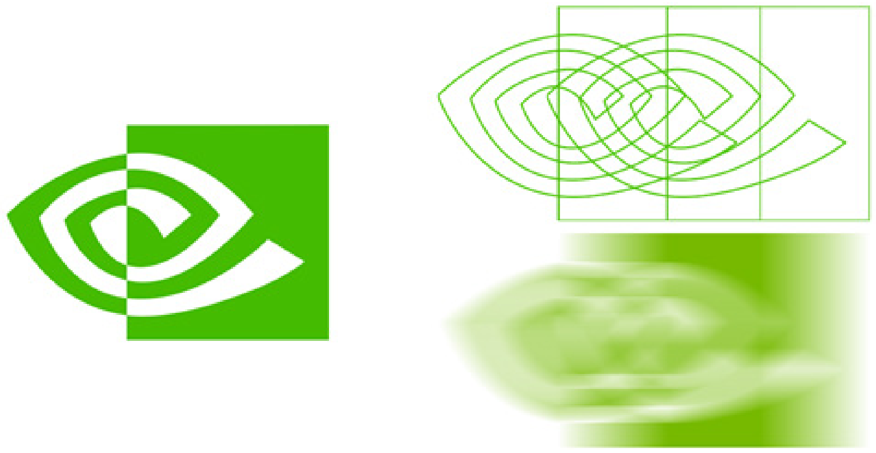


Figure 34. Perceived Blur Due to Object Motion Combined with Retinal and Display Persistence

In Figure 34, the image on the left is what is being displayed on the screen. If this image moves from left to right, it will look to the eye like it is jumping back and forth. The eye integrates this motion and ultimately sees the image on the lower right, a lower resolution version of the original.

The main implication of this phenomenon is that when objects are moving rapidly in the scene, it is wasteful to shade them at full resolution. It would be more efficient to shade at a reduced sampling rate, while still at a high enough rate to be visually equivalent. The savings from optimized shading can be used to deliver a higher frame rate so that the scene is easier to follow.

VRS gives us the tools to do this optimization. In the simplest approach, we can use the motion vectors from Temporal AA to understand motion. The direction and magnitude of motion can be used to directly select an appropriate shading rate per tile.

A related approach would be to use VRS to take advantage of blur effects in applications, where both motion blur and depth of field (DOF) are sometimes explicitly rendered. An application can directly compute the degree and direction of blur of individual objects and use the extent of blur to set a per-triangle shading rate.

Note that the methods of these two examples (Content Adaptive Shading and Motion Adaptive Shading) can also be used in combination, with the final shading rate for a region/triangle computed as an application-specified function of the two rates.

Foveated Rendering

The third example application is Foveated Rendering. Foveated Rendering is based on the observation that the resolution that our eye can perceive depends on viewing angle. We have maximum visual resolution for objects in the center of our field of view, but much lower visual resolution for objects in the periphery. Therefore, if the viewer's eye position is known (via eye tracking in either a VR or non-VR system), this can be used to adjust shading rates appropriately. We can shade at lower rates in the periphery, and higher rates in the center of the field of view.

While these three applications are good examples, we expect that developers discover other innovative applications beyond what has been described here. VRS gives developers advanced control over shading rate, that they are now able to connect to any algorithm that can take advantage of this capability.

Finally, VRS also allows the developer to increase shading rate. When using Multisample Anti-Aliasing (MSAA), the developer may use VRS to increase shading rate from the once per pixel baseline, to 2, 4, or 8 times per pixel. The rate increase can be no more than the MSAA sample count.

TEXTURE SPACE SHADING

Turing GPUs introduce a new shading capability called *Texture Space Shading* (TSS), where shading values are dynamically computed and stored in a texture as texels in a texture space. Later, pixels are texture mapped, where pixels in screen-space are mapped into texture space, and the corresponding texels are sampled and filtered using a standard texture lookup operation. With this technology we can sample visibility and appearance at completely independent rates, and in separate (decoupled) coordinate systems. Using TSS, a developer can simultaneously improve quality and performance by (re)using shading computations done in a decoupled shading space.

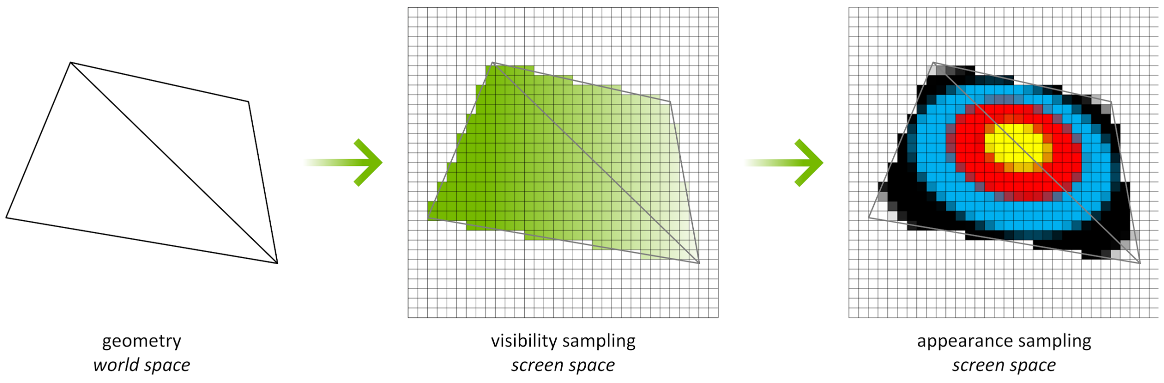
Developers can use TSS to exploit both spatial and temporal rendering redundancy. By decoupling shading from the screen-space pixel grid, TSS can achieve a high-level of frame-to-frame stability, because shading locations do not move between one frame and the next. This temporal stability is important to applications like VR that require greatly improved image quality, free of aliasing artifacts and temporal shimmer.

TSS has intrinsic multi-resolution flexibility, inherited from texture mapping's MIP-map hierarchy, or image pyramid. When shading for a pixel, the developer can adjust the mapping into texture space, which MIP level (level of detail) is selected, and consequently exert fine control over shading rate. Because texels at low levels of detail are larger, they cover larger parts of an object and possibly multiple pixels.

TSS remembers which texels have been shaded and only shades those that have been newly requested. Texels shaded and recorded can be reused to service other shade requests in the same frame, in an adjacent scene, or in a subsequent frame. By controlling the shading rate and reusing previously shaded texels, a developer can manage frame rendering times, and stay within the fixed time budget of applications like VR and AR. Developers can use the same mechanisms to lower shading rate for phenomena that are known to be low frequency, like fog. The usefulness of remembering shading results extends to vertex and compute shaders, and general computations. The TSS infrastructure can be used to remember and reuse the results of any complex computation.

The Mechanics of TSS

Figure 35 illustrates the traditional rasterization and shading process. A 3D scene is rasterized and converted to pixels in screen space. The pixels are tested for visibility, shaded for appearance, and depth-tested. The operations all take place on the same screen-space pixel grid, on the same pixel.



3D scene is rasterized and converted to pixels in screen space and visible pixels are determined and shaded.

Figure 35. Traditional Rasterization and Shading Process

With TSS, the two major operations of visibility sampling (rasterization and z-testing) and appearance sampling (shading) can be decoupled and performed at a different rate, on a different sampling grid, or even on a different timeline. The shading process is no longer tied directly to screen-space pixels, it happens in texture space. In Figure 36, the geometry is still rasterized to produce screen-space pixels, and the visibility test still takes place in screen-space. However, instead of shading in screen-space, texels are found that are required to cover an output pixel. In other words, the footprint of the screen-space pixel is mapped into a separate texture space and shade the associated texels in texture space. The mapping to texture space is a standard texture mapping operation with the same control over the LOD and things like anisotropic filtering. To produce the final screen-space pixels we sample from the shaded texture. The texture is created on-demand based on sample requests, only generating values for texels that are referenced.

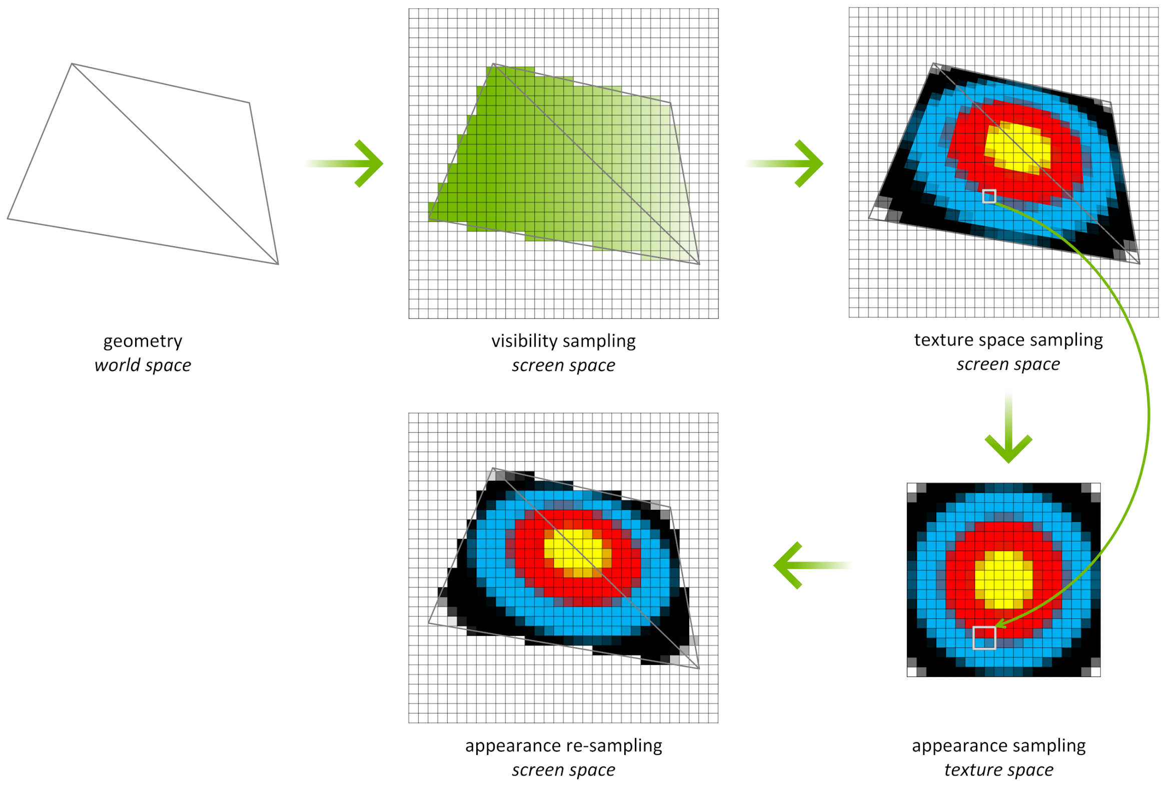


Figure 36. Texture Space Shading Process

One example use case for TSS is improving the efficiency of VR rendering. Figure 35 shows an example use case for TSS in VR rendering. In VR, a stereo pair of images is rendered, with almost all of the elements visible in the left eye also showing up in the right eye view. With TSS, we can shade the full left-eye view, and then render the right eye view by sampling from the completed left-eye view. The right eye view only has to shade new texels in the case that no valid sample was found (for example a background object that was obscured from view from the left-eye perspective but is visible to the right eye).

As mentioned, with TSS, per-pixel shading rate can be dynamically and continuously controlled by adjusting texture LOD. By varying LOD we can select different texture MIP levels as needed to reduce the number of texels shaded. Note that this means that the sampling approach of TSS can also be used to implement many of the same shading rate reduction techniques that are supported by the VRS feature (see the *Variable Rate Shading* section on page 42). Which method is best for the developer depends on their objectives. VRS is a lighter weight change to the rendering pipeline, while TSS has more flexibility and supports additional use cases.

Figure 37. Texture Space Shading for Stereo

MULTI-VIEW RENDERING

Multi-View Rendering MVR) allows developers to efficiently draw a scene from multiple viewpoints or even draw multiple instances of a character in varying poses, all in a single pass. Turing hardware supports up to four views per pass, and up to 32 views are supported at the API level. By fetching and shading geometry only once, Turing optimally processes triangles and their associated vertex attributes while rendering multiple versions. When accessed via the D3D12 View Instancing API, the developer simply uses the variable `SV_ViewID` to index different transformation matrices, reference different blend weights, or control any shader behavior they like, that varies depending on which view they are processing.

With multiple active views, each triangle can have a mix of view-dependent attributes and view-independent attributes (values that are shared across all views). A simple example of a view-dependent attribute is reflection direction, because it depends on the eye's position, vertex position, and normal vector. To improve efficiency, the NVIDIA compiler analyzes the input shader and produce a compiled output that executes view independent code once, with the result shared across all output views, while view dependent attributes are necessarily computed once per output view.

Turing's MVR is an expansion of the Simultaneous Multi-Projection (SMP) functionality introduced in the Pascal architecture. SMP was designed specifically to accelerate stereo and surround rendering cases. With SMP the developer can specify two views, where view dependent attributes are limited to the vertex X coordinate and viewport(s) used for rasterization. Each view can then be multi-cast to a set of up to 16 pre-configured projections (or viewports) to support use cases such as Lens Matched Shading.

Turing removes the limitations on allowed view dependent attributes and increases the number of views supported, while continuing to support up to 16 projections per view. Refer to the *GeForce GTX 1080 Whitepaper* for an in-depth explanation of SMP's capabilities and use cases.

Multi-View Rendering Use Cases

One of the most obvious uses of MVR is an extension of Pascal's Single Pass Stereo (SPS) feature used to accelerate Virtual Reality (VR) rendering. The original SPS allowed for eyes to only be horizontally offset from each other with the same direction of projection. This kind of Head Mounted Display (HMD) configuration is both common and logical, a person's face is quite symmetric, and HMDs overwhelmingly use a single projection plane. Many HMDs use a single physical display for both eyes. However, higher quality HMDs, and newer devices with a very large field of view (FOV), require greater view flexibility to harvest the redundant geometry processing still available in the VR workload.

Figure 38 illustrates the configuration of a 200° FOV HMD where two canted panels are used and require MVR's greater expressive power. MVR's flexibility is also beneficial to support more accurate calibration of standard stereo VR displays, to align to an individual user's face. The simple assumption in stereo rendering that eyes are just offset in X from each other is not quite right, in practice there are some additional asymmetries that require independent projections for the highest fidelity alignment.

Figure 39 shows additional examples of MVR Single-pass rendering of four shadow depth buffers (top left). A pair of characters rendered from the same mesh (top right), where the mesh is fetched once, and the view ID is used to control the generation of the two instances in a single pass. Single pass cascaded shadow map rendering is shown across the bottom of Figure 39.

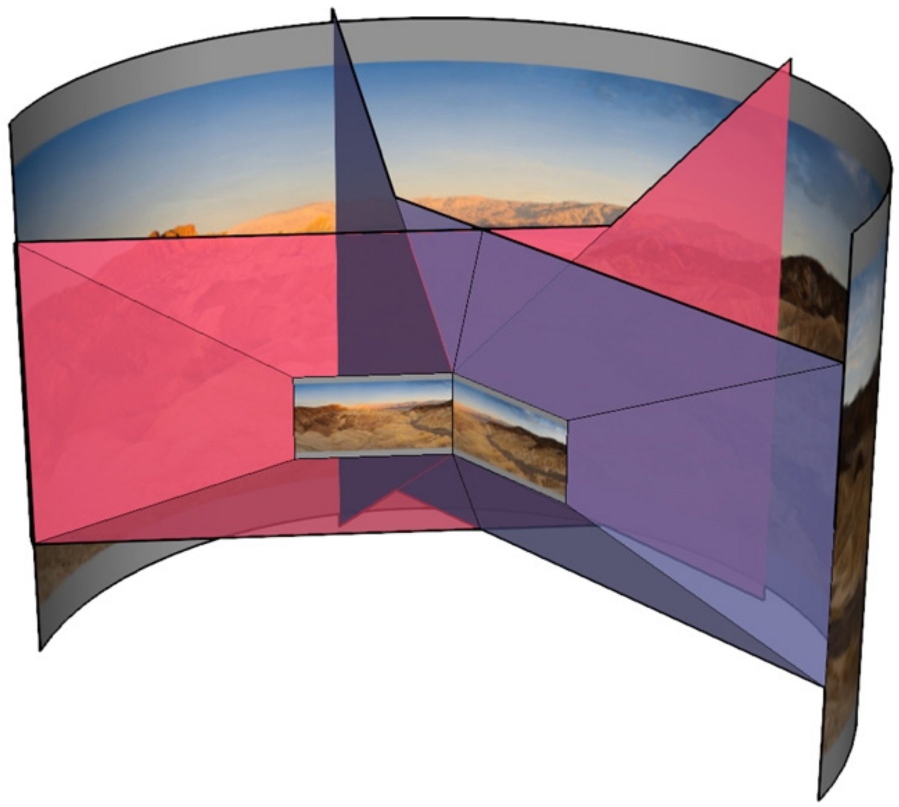


Figure 38. 200° FOV HMD Where Two Canted Panels are Used and Benefit from MVR

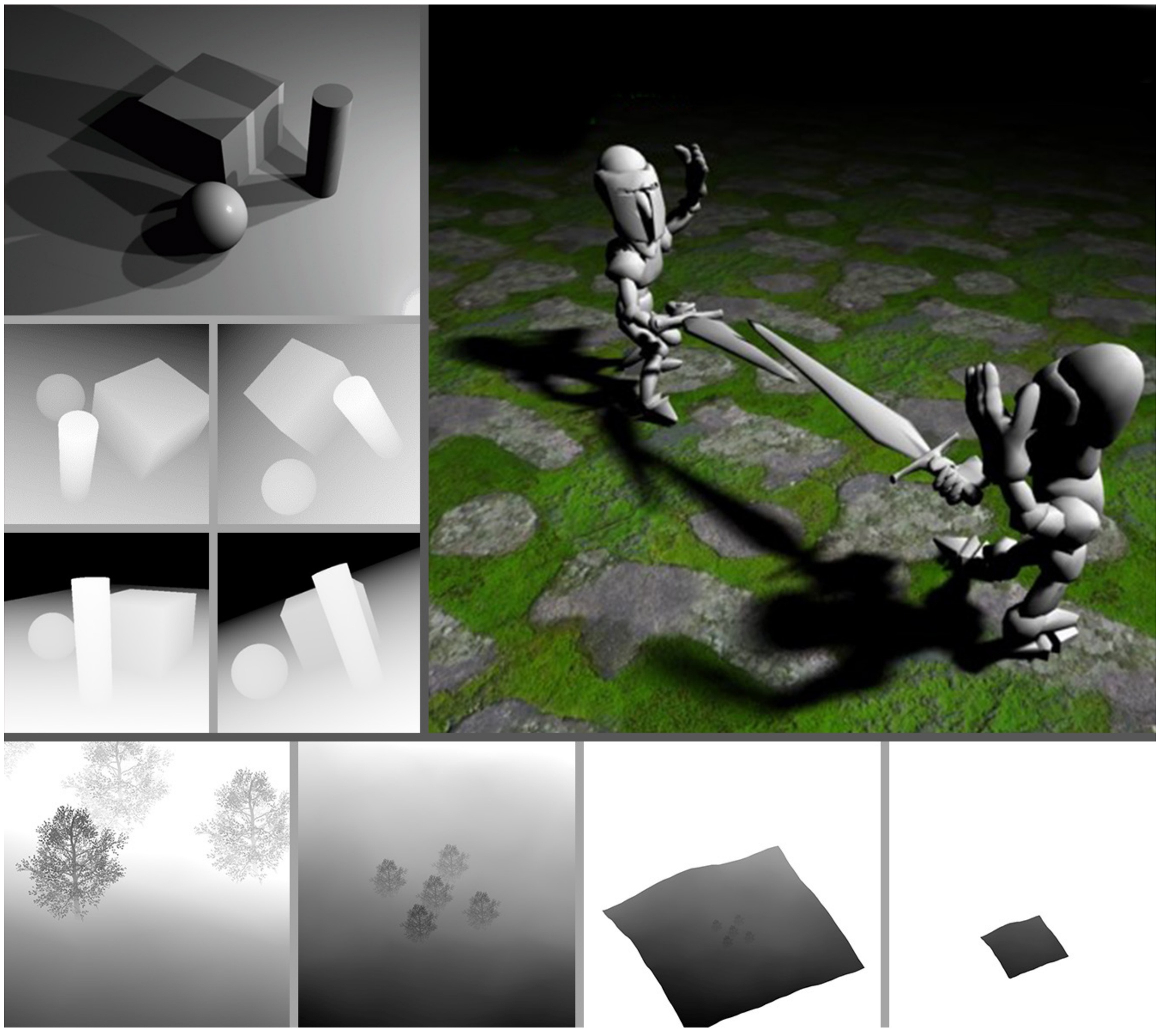


Figure 39. MVR Single Pass Cascaded Shadow Map Rendering

RESOURCE MANAGEMENT AND BINDING MODEL

DX12 introduced the ability to allow resource views to be directly accessed by shader programs without requiring an explicit resource binding step. Turing extends our resource support to include bindless Constant Buffer Views and Unordered Access Views, as defined in Tier 3 of *DX12's Resource Binding Specification*.

Turing's more flexible memory model also allows for multiple different resource types (such as textures and vertex buffers) to be co-located within the same heap, simplifying aspects of memory management for the app. Turing supports Tier 2 of resource heaps.

TURING FEATURES ENHANCE VIRTUAL REALITY

Turing GPU architecture includes many significant advancements for Virtual Reality (VR) technology and Head Mounted Displays (HMD). New Turing ray tracing, shading, and interface technologies improve VR performance, immersion, and comfort. Though expanded upon elsewhere in this document, the Turing advancements for VR are collected here for ease of reference.

Turing GPUs are designed with hardware support for USB Type-C and VirtualLink, the new open industry standard that delivers the power, display, and data required to power VR headsets through a single USB-C connector. Turing GPUs drastically reduce VR connectivity complications, and the single-connector solution enables VR on small form factor devices that traditionally do not support multiple connectors.

Another Turing GPU feature that improves the VR experience is Multi-View Rendering (MVR). MVR is an expansion of the SMP functionality introduced in the Pascal architecture that processes a single geometry stream across two different projection centers to more efficiently render stereo displays for VR. Turing MVR expands the number of viewpoint projections from two to four, enabling headset manufacturers to use the additional viewpoint projections for canted wraparound side views, increasing the quality of its immersion.

Figure 40 shows the Turing features for VR.

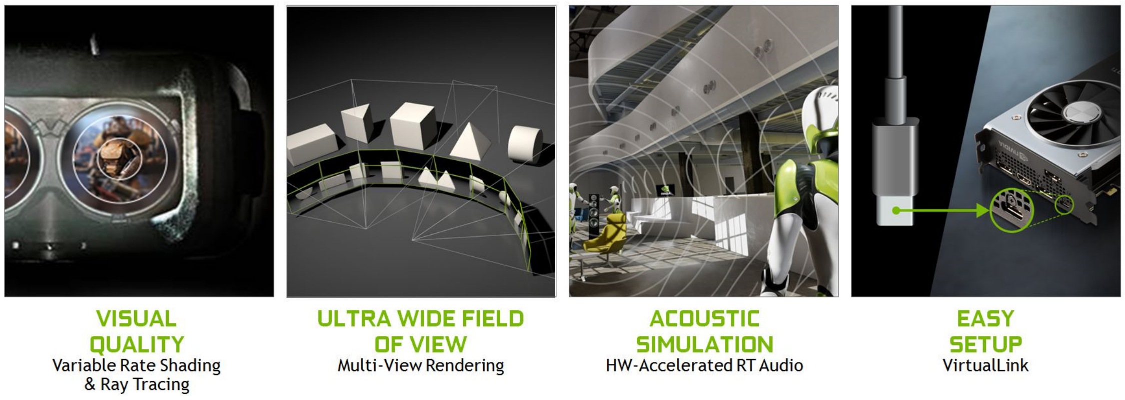


Figure 40. Turing Features for VR

Foveated Rendering is a VR use-case of the new Turing VRS feature. VRS allows developers to control shading rate to exploit phenomena like foveation and object motion for reduced or additional shading, depending on whether efficiency or added detail is desired. Foveated Rendering in VR uses variable shading to reduce shading where the eye is not looking, and even increase shading in the areas where the eye does gaze. VRS gives developers new methods of increasing immersive details and tailoring efficiency on VR.

VR immersion relies on more than just images. Three-dimensional sound is crucial for VR immersion. All games feature 3D sound through simple positioning of direct sound. NVIDIA VRWorks™ Audio (introduced with the Pascal GPU architecture) provided binaural sound elements using the NVIDIA OptiX™ software ray tracing engine to better simulate indirect sound. Sound can bounce off surfaces and arrive at the listener later than direct sound paths, providing reverberation that can represent different types of virtual environments. Turing expands on Pascal's NVIDIA VRWorks Audio by employing RT Cores to accelerate the ray-traced NVIDIA VRWorks Audio by up to 6x.

CONCLUSION

Graphics has just been reinvented. The new NVIDIA Turing GPU architecture is the most advanced and efficient GPU architecture ever built. Turing implements a new Hybrid Rendering model that combines real-time ray tracing, rasterization, AI, and simulation. Teamed with the next generation graphics APIs, Turing enables massive performance gains and incredibly realistic graphics for PC games and professional applications.

APPENDIX A
TURING TU104 GPU

Launching alongside the Turing TU102 GPU is the Turing TU104. The TU104 GPU incorporates all of the new Turing features found in TU102, including the RT Cores, Turing Tensor Cores, and the architectural changes made to the Turing SM.

The full TU104 chip contains six GPCs, 48 SMs, and eight 32-bit memory controllers (256-bit total). In TU104, each GPC includes a raster unit and four TPCs. Each TPC contains a PolyMorph Engine and two SMs.

Each SM includes the new RT Core. Like TU102, each SM also includes 64 CUDA Cores, 256 KB register file, 96 KB L1 data cache/shared memory cache, and four texture units. The full TU104 chip contains 13.6 Billion transistors and includes 3072 CUDA Cores, 368 Tensor Cores, and 48 RT Cores. TU104 also supports second-generation NVLink. One x8 NVLink link is included, providing 25 GB/sec of bandwidth in each direction (50 GB/sec total bandwidth). Figure 41 shows the Turing TU104 full-chip diagram.

The TU104 GPU will be used in different levels of GeForce, Tesla, and Quadro products, such as the GeForce RTX 2080, Tesla T4, and Quadro RTX 5000.

Table 4 lists the specification comparison of the GeForce RTX 2080 and Quadro RTX 5000.

Table 5 lists the specifications of Tesla T4.

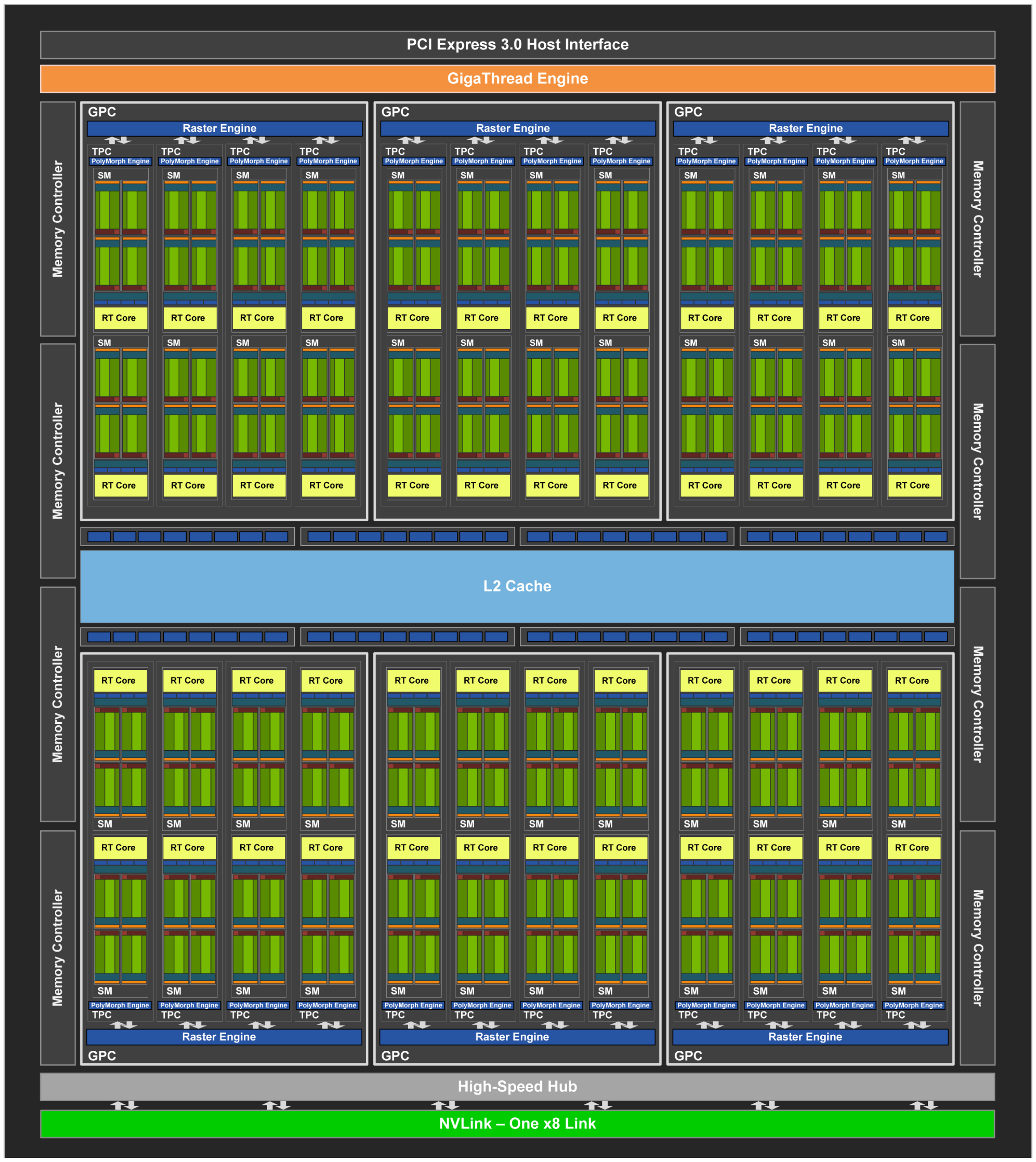


Figure 41. Turing TU104 Full Chip Diagram

Table 4. Comparison of NVIDIA Pascal GP104 and Turing TU104 GPUs

GPU Features	GeForce GTX 1080	GeForce RTX 2080	Quadro P5000	Quadro RTX 5000
Architecture	Pascal	Turing	Pascal	Turing
GPCs	4	6	4	6
TPCs	20	23	20	24
SMs	20	46	20	48
CUDA Cores / SM	128	64	128	64
CUDA Cores / GPU	2560	2944	2560	3072
Tensor Cores / SM	NA	8	NA	8
Tensor Cores / GPU	NA	368	NA	384
RT Cores	NA	46	NA	48
GPU Base Clock MHz (Reference / Founders Edition)	1607 / 1607	1515 / 1515	1607	1620
GPU Boost Clock MHz (Reference / Founders Edition)	1733 / 1733	1710 / 1800	1733	1815
RTX-OPS (Tera-OPS) (Reference / Founders Edition)	8.9 / 8.9	57 / 60	NA	62
Rays Cast (Giga Rays/sec) (Reference / Founders Edition)	0.89	8 / 8	NA	8
Peak FP32 TFLOPS* (Reference / Founders Edition)	8.9	10 / 10.6	8.9	11.2
Peak INT32 TIPS* (Reference/Founders Edition)	NA	10 / 10.6	NA	11.2
Peak FP16 TFLOPS* (Reference / Founders Edition)	NA	20.1 / 21.2	NA	22.3
Peak FP16 Tensor TFLOPS with FP16 Accumulate* (Reference/Founders Edition)	NA	80.5 / 84.8	NA	89.2
Peak FP16 Tensor TFLOPS with FP32 Accumulate* (Reference/Founders Edition)	NA	40.3 / 42.4	NA	89.2
Peak INT8 Tensor TOPS* (Reference / Founders Edition)	NA	161.1 / 169.6	NA	178.4
Peak INT4 Tensor TOPS* (Reference / Founders Edition)	NA	322.2 / 339.1	NA	356.8
Frame Buffer Memory Size and Type	8192 MB GDDR5X	8192 MB GDDR6	16384 GDDR5X	16384 GDDR6
Memory Interface	256-bit	256-bit	256-bit	256-bit
Memory Clock (Data Rate)	10 Gbps	14 Gbps	9 Gbps	14 Gbps
Memory Bandwidth (GB/sec)	320	448	288	448
ROPs	64	64	64	64
Texture Units	160	184	160	192
Texel Fill-rate (Gigatexels/sec)	277.3 / 277.3	314.6 / 331.2	277	348
L2 Cache Size	2048 KB	4096 KB	2048 KB	4096 KB
Register File Size/SM	256 KB	256 KB	256 KB	256 KB

GPU Features	GeForce GTX 1080	GeForce RTX 2080	Quadro P5000	Quadro RTX 5000
Register File Size/GPU	5120 KB	11776 KB	5120 KB	12288 KB
TDP⋆ (Reference / Founders Edition)	180 / 180 W	215 / 225 W	180 W	230 W
Transistor Count	7.2 Billion	13.6 Billion	7.2 Billion	13.6 Billion
Die Size	314 mm²	545 mm²	314 mm²	545 mm²
Manufacturing Process	16 nm	12 nm FFN	16 nm	12 nm FFN
Note: * Peak TFLOPS and TOPS rates are based on GPU Boost Clock. ⋆ Power figure represents Graphics Card TDP only. The use of the VirtualLink™/USB Type-C™ connector requires up to an additional 35 W of power that is not represented in this power figure.				

The NVIDIA Tesla T4 is the first Turing-based GPU designed for inferencing applications in the Data Center, enterprise and edge devices. The TU104 chip used for the Tesla T4 includes five GPCs, 20 TPCs, 40 SMs, a total of 2,560 CUDA Cores, and 320 Turing Tensor Cores. The Tesla T4 TU104 chip also includes a 256-bit memory interface and a 10 Gbps memory data rate for a total bandwidth of 320 GB/s (see Table 5 for a comparison of the Pascal Tesla P4 and the Turing Tesla T4).

Table 5. Comparison of the Pascal Tesla P4 and the Turing Tesla T4

GPU	Tesla P4 (Pascal)	Tesla T4 (Turing)
GPCs	4	5
TPCs	20	20
SMs	20	40
CUDA Cores/SM	128	64
CUDA Cores/GPU	2,560	2,560
Tensor Cores/SM	NA	8
Tensor Cores/GPU	NA	320
RT Cores	NA	40
GPU Base Clock MHz	810	585*
GPU Boost Clock MHz	1,063	1,590
Peak FP32 TFLOPS	5.5	8.1
Peak INT32 TIPS	NA	8.1
Peak FP16 TFLOPS	NA	16.2
Peak FP16 Tensor TFLOPS with FP16 Accumulate*	NA	65
Peak FP16 Tensor TFLOPS with FP32 Accumulate*	NA	65
Peak INT8 Tensor TOPS*	22	130
Peak INT4 Tensor TOPS*	NA	260
Frame Buffer Memory Size and Type	8192 MB GDDR5X	16384 MB GDDR6
Memory Interface	256-bit	256-bit
Memory Clock (Data Rate)	6 Gbps	10 Gbps
Memory Bandwidth (GB/sec)	192	320
ROPs	64	64
TDP	75 Watts	70 Watts

GPU	Tesla P4 (Pascal)	Tesla T4 (Turing)
Transistor Count	7.2 Billion	13.6 Billion
Die Size	314	545
Manufacturing Process	16 nm	12 nm FFN
Note: * Peak TFLOPS and TOPS rates are based on GPU Boost Clock. * The Tesla T4 Base Clock is designed for a 70W TDP and operating efficiently in data center server racks. While the T4 can operate at much higher clocks for many workloads as seen by its high Boost Clock, its Base Clock indicates the lowest clock that would typically be seen in very stressful inferencing workloads.		

APPENDIX B TURING TU106 GPU

The Turing TU106 GPU, used in the GeForce RTX 2070, ships in October 2018. The GeForce RTX 2070 is designed to deliver the best performance and energy efficiency in its class. Most of the key new features found in the Turing architecture are also supported by TU106, including the RT Cores, Turing Tensor Cores, and all of the architectural changes made to the Turing SM. Compared to TU102 and TU104, TU106 does not offer NVLink or SLI support.

The GeForce RTX 2070 is based on the full implementation of the TU106 GPU, which contains three GPCs, 36 SMs, and eight 32-bit memory controllers (256-bit total). In the TU106, each GPC includes a raster unit and six TPCs. Each TPC contains a PolyMorph Engine and two SMs. Figure 42 shows the Turing TU106 full-chip diagram.

Like TU102 and TU104, each SM in TU106 includes the new RT Core for raytracing. Each SM also includes 64 CUDA Cores, 256 KB register file, 96 KB L1 data cache/shared memory cache, and four texture units. The full TU106 GPU contains 10.8 Billion transistors and includes 2304 CUDA Cores, 288 Tensor Cores, and 36 RT Cores. Table 6 contains the comparison of the NVIDIA Pascal GP104 to Turing TU106.

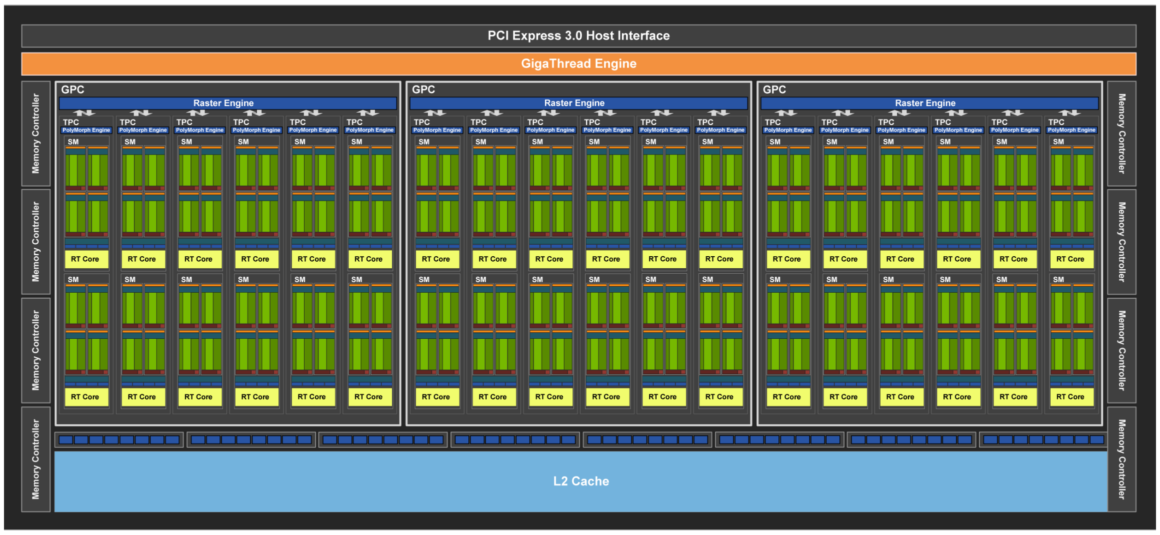


Figure 42. Turing TU106 Full Chip Diagram

Table 6. Comparison of NVIDIA Pascal GP104 to Turing TU106 GPUs

GPU Features	GeForce GTX 1070 (GP104)	GeForce RTX 2070 (TU106)
Architecture	Pascal	Turing
GPCs	3	3
TPCs	15	18
SMs	15	36
CUDA Cores / SM	128	64
CUDA Cores / GPU	1920	2304
Tensor Cores / SM	NA	8
Tensor Cores / GPU	NA	288
RT Cores	NA	36
GPU Base Clock MHz (Reference / Founders Edition)	1506 / 1506	1410 / 1410
GPU Boost Clock MHz (Reference / Founders Edition)	1683 / 1683	1620 / 1710
RTX-OPS (Tera-OPS) (Reference / Founders Edition)	6.5 / 6.5	42 / 45
Rays Cast (Giga Rays/sec) (Reference / Founders Edition)	.065 / .065	6 / 6
Peak FP32 TFLOPS* (Reference / Founders Edition)	6.5 / 6.5	7.5 / 7.9
Peak INT32 TIPS* (Reference/Founders Edition)	NA	7.5 / 7.9
Peak FP16 TFLOPS* (Reference / Founders Edition)	NA	14.9 / 15.8
Peak FP16 Tensor TFLOPS with FP16 Accumulate* (Reference/Founders Edition)	NA	59.7 / 63

GPU Features	GeForce GTX 1070 (GP104)	GeForce RTX 2070 (TU106)
Peak FP16 Tensor TFLOPS with FP32 Accumulate* (Reference/Founders Edition)	NA	29.9 / 31.5
Peak INT8 Tensor TOPS* (Reference / Founders Edition)	NA	119.4 / 126
Peak INT4 Tensor TOPS* (Reference / Founders Edition)	NA	238.9 / 252.1
Frame Buffer Memory Size and Type	8192 MB GDDR5	8192 MB GDDR6
Memory Interface	256-bit	256-bit
Memory Clock (Data Rate)	8 Gbps	14 Gbps
Memory Bandwidth (GB/sec)	256	448
ROPs	64	64
Texture Units	120	144
Texel Fill-rate (Gigatexels/sec)	202 / 202	233.3 / 246.2
L2 Cache Size	2048 KB	4096 KB
Register File Size/SM	256 KB	256 KB
Register File Size/GPU	3840 KB	9216 KB
TDP (Reference / Founders Edition)	150 / 150 Watts	175 / 185 Watts
Transistor Count	7.2 Billion	10.8 Billion
Die Size	314 mm²	445 mm²
Manufacturing Process	16 nm	12 nm FFN
*Note: Peak TFLOPS and TOPS rates are based on GPU Boost Clock.		

APPENDIX C
RTX-OPS DESCRIPTION

THE HYBRID RENDERING MODEL

Previously, real-time graphics relied on rasterizing triangles to render images. Now, with the introduction of RT Cores and Tensor Cores, Turing hardware enables real-time ray tracing for lighting and the use of AI for image enhancement and other applications. The graphics API has evolved in the same direction, with the introduction of DirectX Raytracing and Windows ML as part of the Windows 10 October 2018 update. Taken together, these changes enable a new rendering model, Hybrid Rendering, in which graphics applications use a combination of traditional rendering, ray traced rendering, and AI to produce amazing images in real time.

Understanding usable operations for hybrid rendering requires an understanding of the workload. Now, there are multiple throughputs that matter. High operation throughput for ray tracing and AI is critical, but neither is used throughout the entire frame time, so just adding up those operations along with shader operations would not produce a useful metric. As a first step, it is important to understand how much time is spent on each of these workloads (see Figure 43).

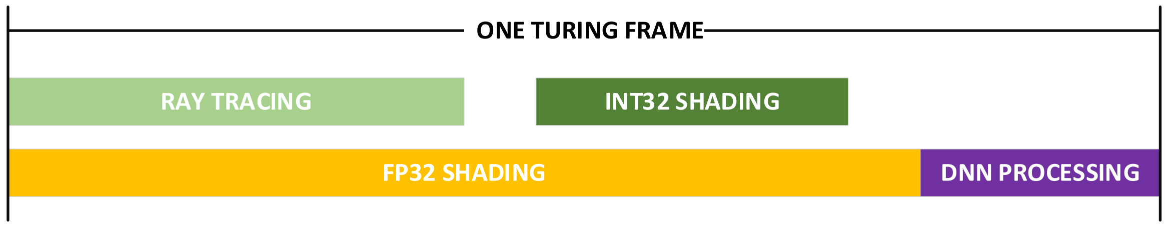


Figure 43. Workload Distribution Over One Turing Frame Time

Figure 43 illustrates an example workload distribution over one frame time, based on measured data from applications running on Turing, in particular:

- Using DLSS as a representative DNN workload (purple), we observe that it takes about 20% of the frame time. The remaining 80% time is doing rendering (yellow).
- Of the remaining rendering time, some time will be spent ray tracing (green) while some time is spent in traditional rasterization or G-Buffer evaluation. The amount of time will vary based on content. Based on the games and demo applications we've evaluated so far, we found that a 50/50-time split is representative. So, in Figure 43, Ray Tracing is about half of the FP32 shading time. In Pascal, ray tracing is emulated in software on CUDA cores, and takes about 10 TFLOPs per Giga Ray, while in Turing this work is performed on the dedicated RT cores, with about 10 Giga Rays of total throughput or 100 tera-ops of compute for ray tracing.
- A third factor to consider for Turing is the introduction of integer execution units that can execute in parallel with the FP32 CUDA cores. Analyzing a breadth of shaders from current games, we found that for every 100 FP32 pipeline instructions there are about 35 additional instructions that run on the integer pipeline. In a single-pipeline architecture, these are instructions that would have had to run serially and take cycles on the CUDA cores, but in the Turing architecture they can now run concurrently. In the timeline above, the integer pipeline is assumed to be active for about 35% of the shading time.

  Given this workload model, it becomes possible to understand the usable ops in Turing and compare vs a previous generation GPU that only had one kind of operation instead of four. This is the purpose of RTX-OPS—to provide a useful, workload-based metric for hybrid rendering workloads.

RTX-OPS WORKLOAD-BASED METRIC EXPLAINED

To compute RTX-OPs, the peak operations of each type based is derated on how often it is used. In particular:

- Tensor operations are used 20% of the time
- CUDA cores are used 80% of the time
- RT cores are used 40% of the time (half of 80%)
- INT32 pipes are used 28% of the time (35% of 80%)

For example, RTX-OPS = TENSOR * 20% + FP32 * 80% + RTOPS * 40% + INT32 * 28%

Figure 44 shows an illustration of the peak operations of each type for RTX 2080 Ti. Plugging in those peak operation counts results in a total RTX-OPs number of 78.
For example, 14 * 80% + 14 * 28% + 100 * 40% + 114 * 20%.

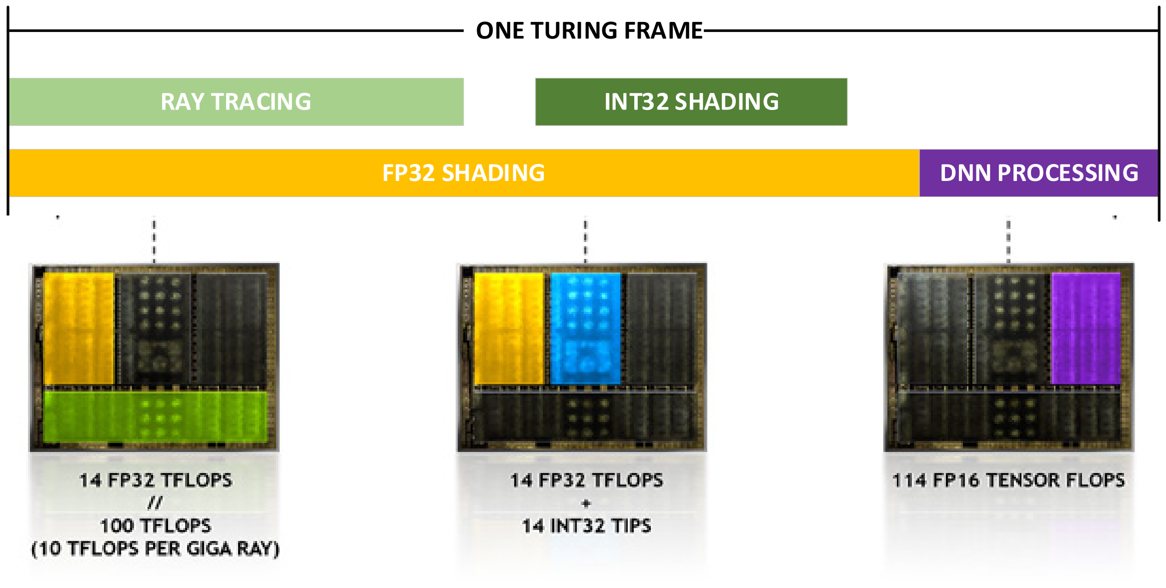


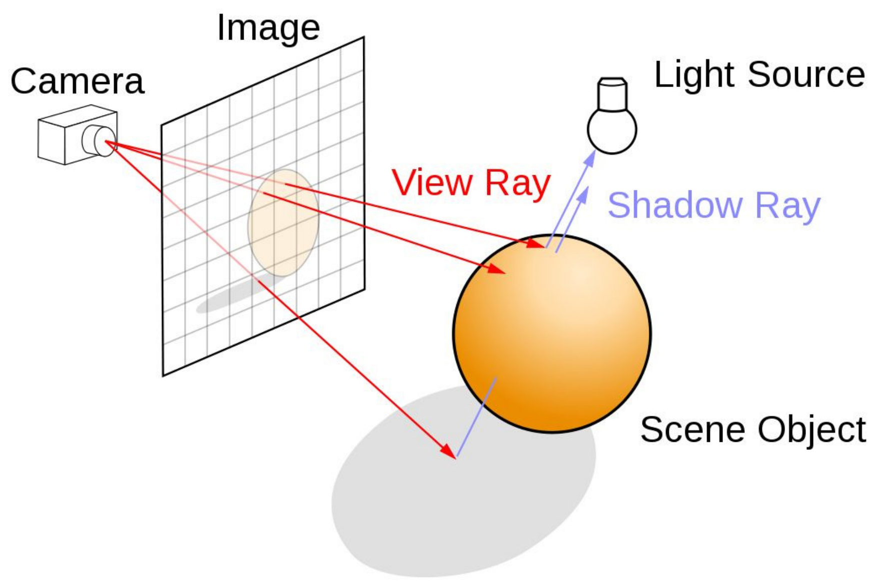
Figure 44. Peak Operations of Each Type Base for RTX 2080 Ti

APPENDIX D
RAY TRACING OVERVIEW

Ray tracing is a rendering technique that can realistically simulate the lighting of a scene and its objects by rendering physically correct reflections, refractions, shadows, and indirect lighting. Many ray tracing algorithms work in reverse of how you might think they should work. Instead of tracing light rays from light sources in the 3D scene to your eyes, rays are actually cast or *shot* backwards from the view camera (which determines your view into the scene) through the 2D viewing plane (pixel plane) out into the 3D scene and back to the light sources. This reverse tracing process is far more efficient than tracing all rays emitted in multiple directions from light sources, because only the rays that pass through the viewing plane and reach your eyes are necessary for rendering a scene. Some rays directly reach your eyes from light sources, others may be blocked by objects in the scene causing shadows, and still others reflect or refract off other objects before reaching your eyes.

When rays shot into the scene intersect objects, the color and lighting information at the points of intersection on object surfaces contribute to the various pixel color and illumination levels. Different objects have different surface properties that can reflect, refract, or absorb light in different ways, and must also be considered. Rays can reflect off objects and hit other objects, or travel through the surfaces of transparent objects before reaching a light source, and the color and lighting information from all the intersected objects may contribute to the final pixel colors.

Figure 45 shows the basic ray tracing process.



Source: https://en.wikipedia.org/wiki/Ray_tracing_(graphics)#/media/File:Ray_trace_diagram.svg

Figure 45. Basic Ray Tracing Process

Ray tracing can be very costly in terms of the computational horsepower required to generate realistic-looking scenes, largely related to the number of rays shot into the scene, and the number of additional rays generated by reflections and refractions. Many factors contribute to the number of rays shot into the scene, including, but not limited to the number and type of objects desired to be ray traced, available GPU processing power per frame, screen resolution, and number of rays desired to be shot through each pixel into the scene.

Ray tracing can produce images that are indistinguishable from those captured by a camera and has been used extensively for movie special effects for years. In fact, live action movies use ray tracing to blend computer-generated effects with images captured by cameras seamlessly, while animated feature films can also look amazingly realistic using ray tracing.

BASIC RAY TRACING MECHANICS

Understanding how ray tracing works at a deeper level requires understanding a few fundamentals, starting with ray casting, which is a visibility determination technique used in the inner loops at the core of photorealistic ray-traced renderers.

Ray casting is actually the process in a ray tracing algorithm that shoots one or more rays from the camera (eye position) through each pixel in an image plane, and then tests to see if the rays intersect any primitives (triangles) in the scene. If a ray passing through a pixel and out into the 3D scene hits a primitive, then the distance along the ray from the origin (camera or eye point) to the primitive is determined, and the color data from the primitive contributes to the final color of the pixel. The ray may bounce and hit other objects and pick up color and lighting information from those other objects. (A related technique called Path Tracing is a far more intensive form of ray tracing that might trace hundreds or thousands of rays through each pixel and follow the rays through numerous bounces off or through objects before reaching the light source in order to collect color and lighting information).

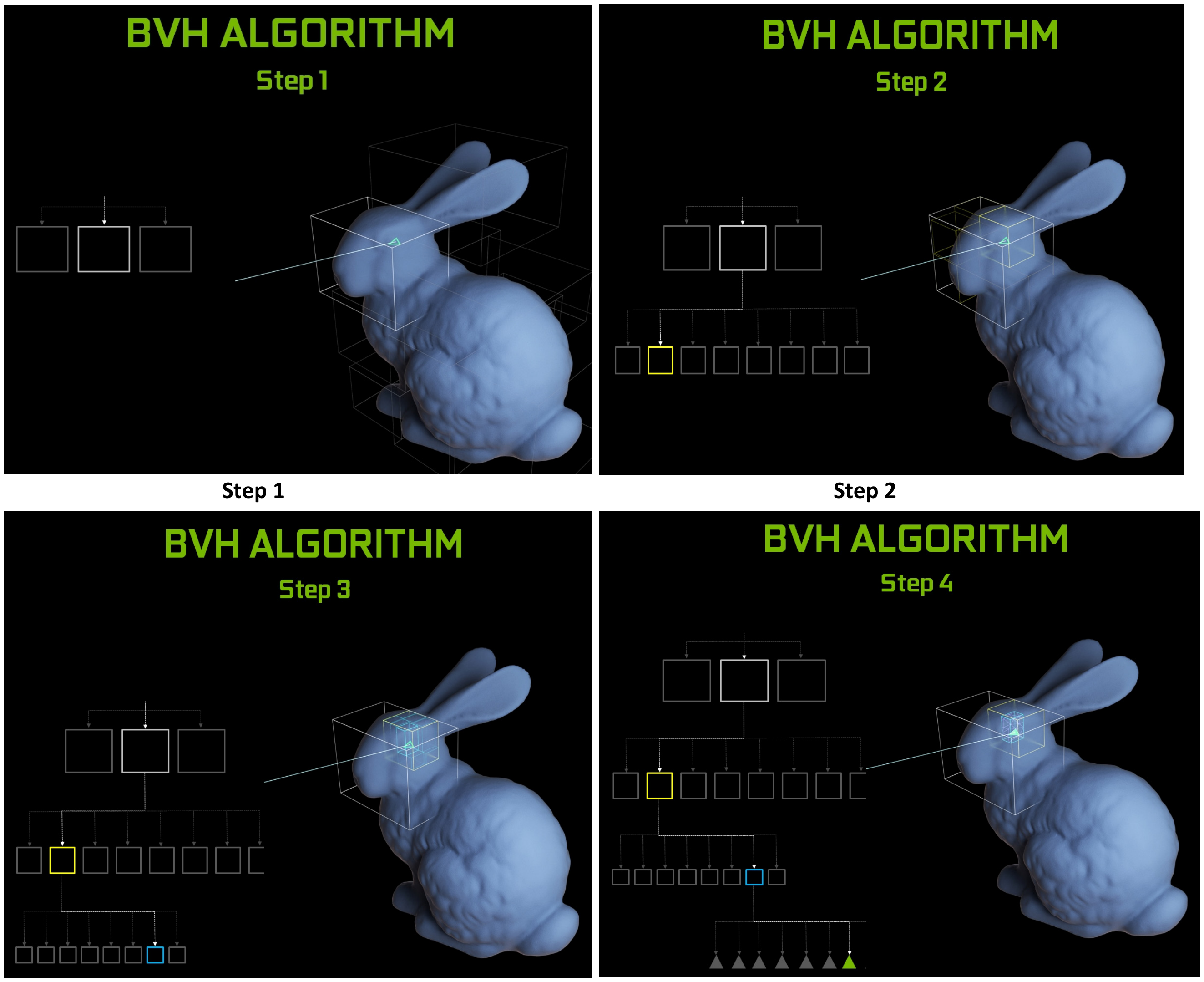
Different types of techniques and optimizations can be used to accelerate ray/primitive (ray/triangle) intersection testing and reduce the number of rays that must be cast to improve performance. Otherwise, testing each ray against every primitive in the scene is incredibly inefficient and computationally very expensive.

Bounding Volume Hierarchy

One popular ray tracing acceleration technique is to use a tree-based *acceleration structure* that contains multiple hierarchically-arranged bounding boxes (bounding volumes) that encompass or surround different amounts of scene geometry. Large outer bounding boxes can encompass many primitives and also a number of increasingly smaller bounding boxes that each surround smaller amounts of geometry. The hierarchically-arranged bounding boxes are aptly called a Bounding Volume Hierarchy or BVH.

A BVH is often arranged into a tree structure with many levels, with one or more nodes per level, starting with a single root node at the top level, and flowing downwards into multiple descendant nodes at different levels. Figure 46 shows how a BVH is represented as a tree structure with larger bounding boxes associated with higher nodes in the tree, and smaller boxes when traversing down the tree. Each node is encompassed by a bounding box that bounds all of its descendant nodes and their bounding boxes. Each ray is tested against the BVH using a depth-first tree traversal process. The process starts by testing the ray against the root node bounding box (see the Stanford Bunny head is completely encapsulated by a big bounding box at the topmost node) and working down the tree of descendant nodes to test which successively smaller bounding boxes are intersected by the ray.

Using a BVH approach to ray/primitive testing significantly reduces the number of tests required. Instead of naively testing rays against each and every primitive in the scene, tests only need to be performed against much fewer numbers of bounding boxes at each level of the tree, until the ray finally hits a leaf node that contains a primitive.



Step 1 **Step 2**

Step 3 **Step 4**

Figure 46. Abstraction of Tree Traversal and a Ray Intersecting Different Levels of Bounding Boxes

Prior to rendering a scene for the first time, a BVH structure must be created (called BVH building) from source geometry. If the next frame has significant changes compared to the prior frame, a new BVH build operation may be required to represent all the changes in the scene. However, in most cases, an existing BVH structure can be modified (called BVH refitting) based on only certain scene changes, without requiring an entirely new BVH build. The refitting procedure is less computationally expensive and is the common case for real world rendering.

DENOISING FILTERING

In addition to acceleration structures that aid in improving ray tracing performance, various advanced filtering techniques can also improve performance and image quality without requiring additional rays to be cast. One such filtering technique is called *denoising*. Denoising can significantly improve the visual quality of noisy images that might be constructed of sparse data, have random artifacts, visible quantization noise, or other types of noise. In fact, many types and causes of image noise exist, and similarly many types of denoising methods also exist. Denoising filtering is especially effective at reducing the time ray-traced images take to render and can produce high fidelity images from ray tracers that appear visually noiseless.

Currently, NVIDIA is making use of both AI-based and non-AI-based algorithms for denoising, choosing whatever is best for a particular application. In the future we expect AI-based denoising to continue to improve and replace non-AI-based methods, repeating the trend that has been seen in many other image-related applications for AI.

RAY-TRACED SHADOWS, AMBIENT OCCLUSION, AND REFLECTIONS

Shadows are a very important visual cue. Shadows help ground objects, and as an integral part of the lighting they set the mood of a scene. Most games today use shadow maps, although a few other techniques have been used to address some shadow map drawbacks. Ray tracing, combined with denoising, allows Turing GPUs to overcome challenges of shadow maps, such as resolution mismatch (which make it difficult to generate hard shadow edges), and contact hardening.

Contact hardening can be approximated with techniques such as Percentage Closer Soft Shadows PCSS) and Distance Field Shadows. While PCSS is quite expensive computationally, it is unable to generate fully correct shadows from arbitrary area lights. Distance Field Shadows are limited to static geometry in current implementations.

With Turing RTX-based ray tracing acceleration and fast denoising algorithms, ray-traced shadows can replace shadow maps, and provide a practical technique to simulate physically correct contact hardening in shadows from all types of area lights.

As shown in Figure 47 and Figure 48, the Shadow Map implementation uniformly blurs the shadow edges a bit but does not provide correct contact hardening. Ray-traced shadows are created from the same directional light, with variable cone angles. The ray-traced shadows can provide completely hard edges if so desired (with a cone angle of 0 degrees on the bottom left), or correct contact hardening for varying cone angles (with 1.5 and 10 degrees cones shown on the right).

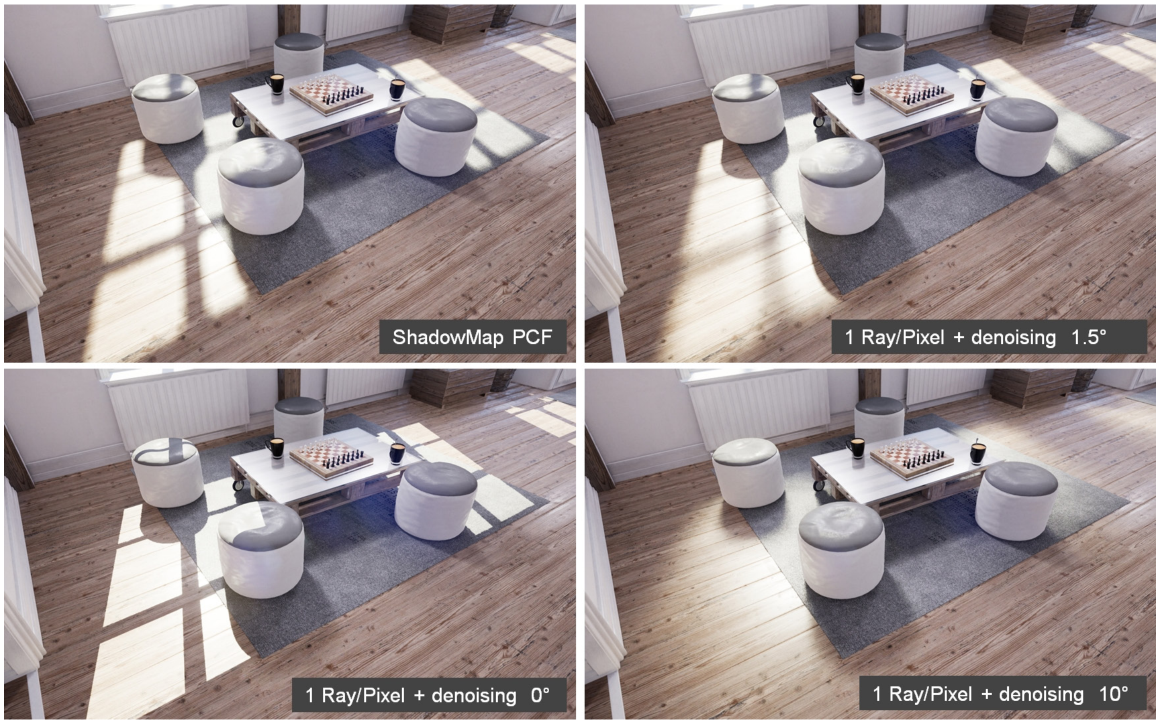

ShadowMap PCF
1 Ray/Pixel + denoising 1.5°
1 Ray/Pixel + denoising 0°
1 Ray/Pixel + denoising 10°

Figure 47. Shadow Map Percentage Closer Filtering (PCF) versus Ray Tracing with Denoising

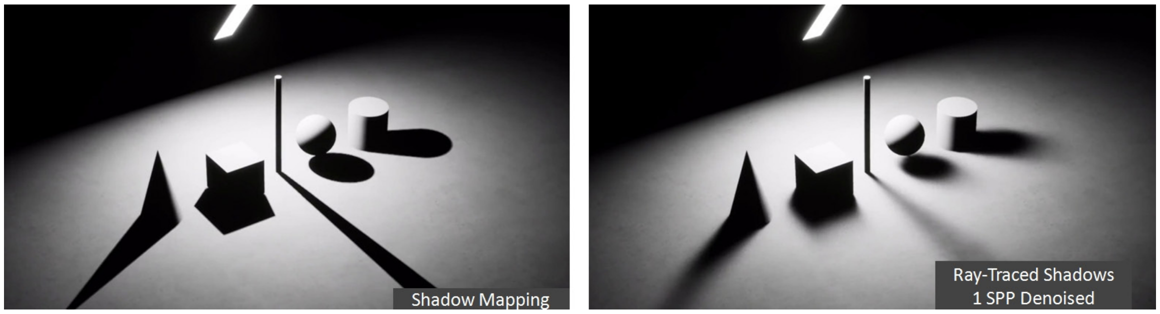
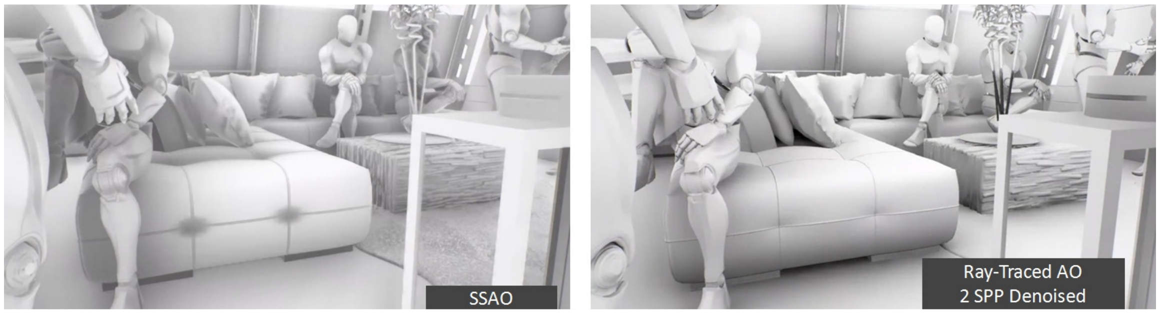


Figure 48. Shadow Mapping Compared to Ray Traced Shadows that use 1 Sample Per Pixel and Denoising

Ambient occlusion (AO), like shadows helps ground objects in their environment. Although considered a hack, AO counters the lack of dynamic global illumination by highlighting creases and geometric complexity in objects, which would otherwise look flat. Ray-traced ambient occlusion has a subtle effect than shadows.

Figure 49 compares the popular Screen-Space Ambient Occlusion (SSAO) technique that has been used for years in real time graphics with Ray-Traced Ambient Occlusion (RTAO).



Screen-Space Ambient Occlusion compared to Ray-Traced Ambient Occlusion with two samples per pixel and denoising applied. Notice far more realistic shadows and definition in the couch, pillows, center table, the characters, and the table to the right.

Figure 49. Screen-Space Ambient Occlusion Compared to Ray-Traced Ambient Occlusion

Ray-traced reflections result in the most obvious visual quality improvements from ray tracing, especially when used in scenes with specular and glossy materials. The most common techniques used today, such as screen-space reflections blended with cubemap probes, have limitations. Screen-space reflections, while cost effective, often results in holes or messy artifacts in the rendered image. Cubemap probes are most often static and low resolution, so they are only an acceptable fallback for glossy materials in scenes with mostly static lighting. Planar reflections are limited by the number that can be afforded to generate with rasterization-based techniques.

Ray-traced reflections, combined with denoising, avoids all these problems and results in artifact-free reflections, including physically correct glossy reflections. Furthermore, since existing high-end GPUs are capable of generating some ray-traced reflections at real-time frame-rates, Turing allows broader use of ray-traced reflections while keeping them affordable.

RTX ray tracing renders physically correct reflections with high visual impact, especially in scenes with many specular (flat) surfaces and glossy materials, as shown in Figure 50.

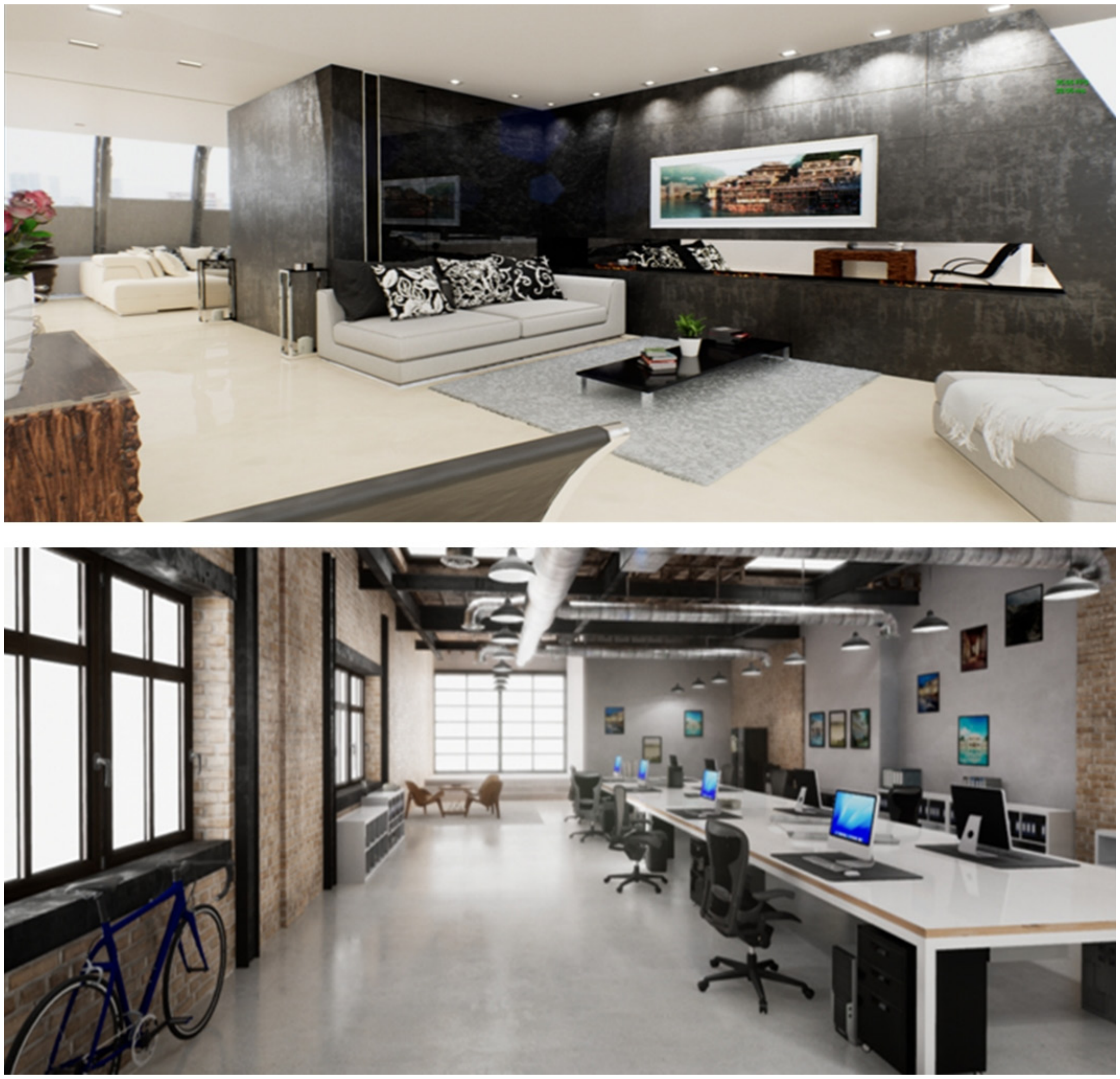
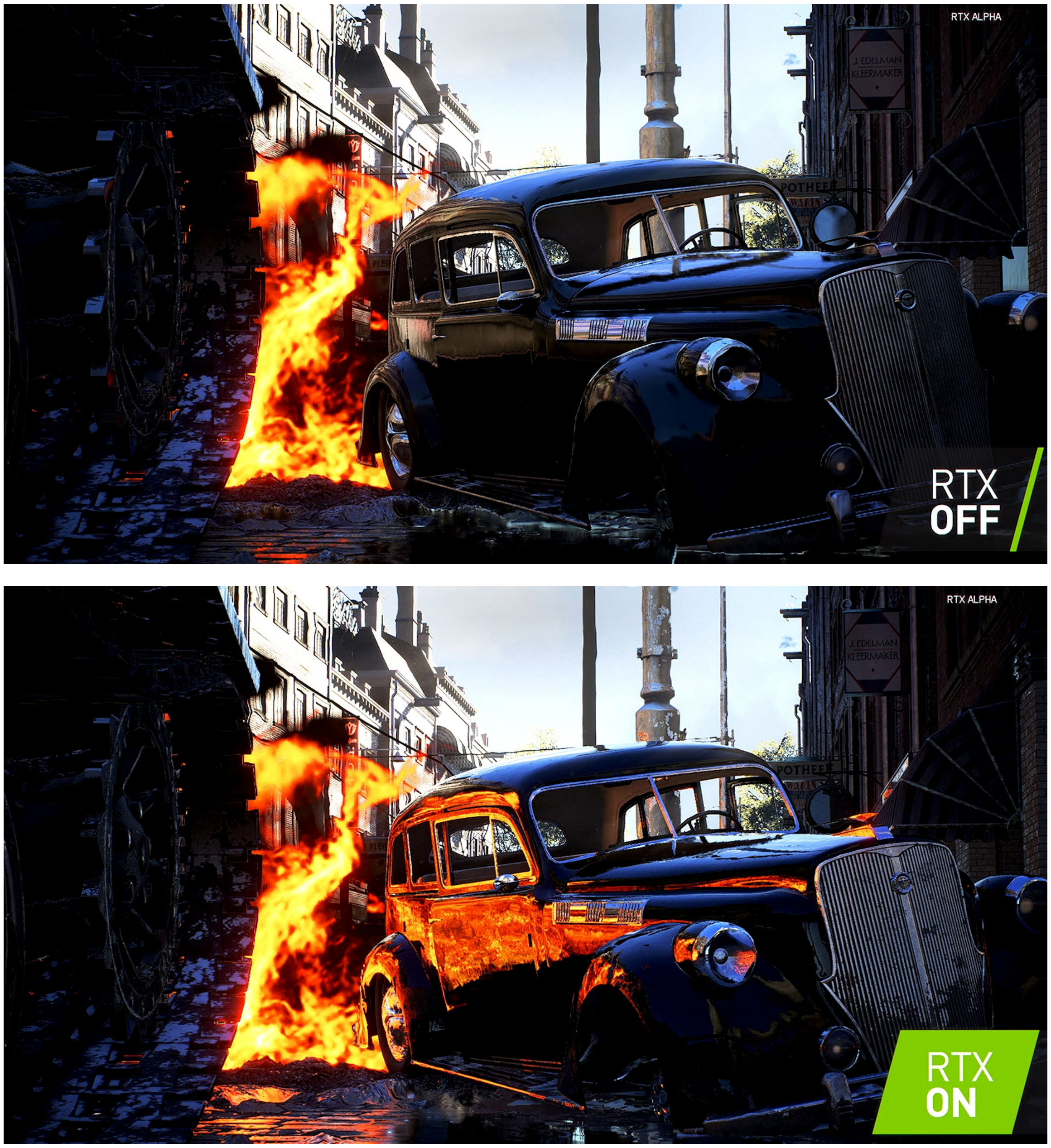


Figure 50. RTX Ray Tracing

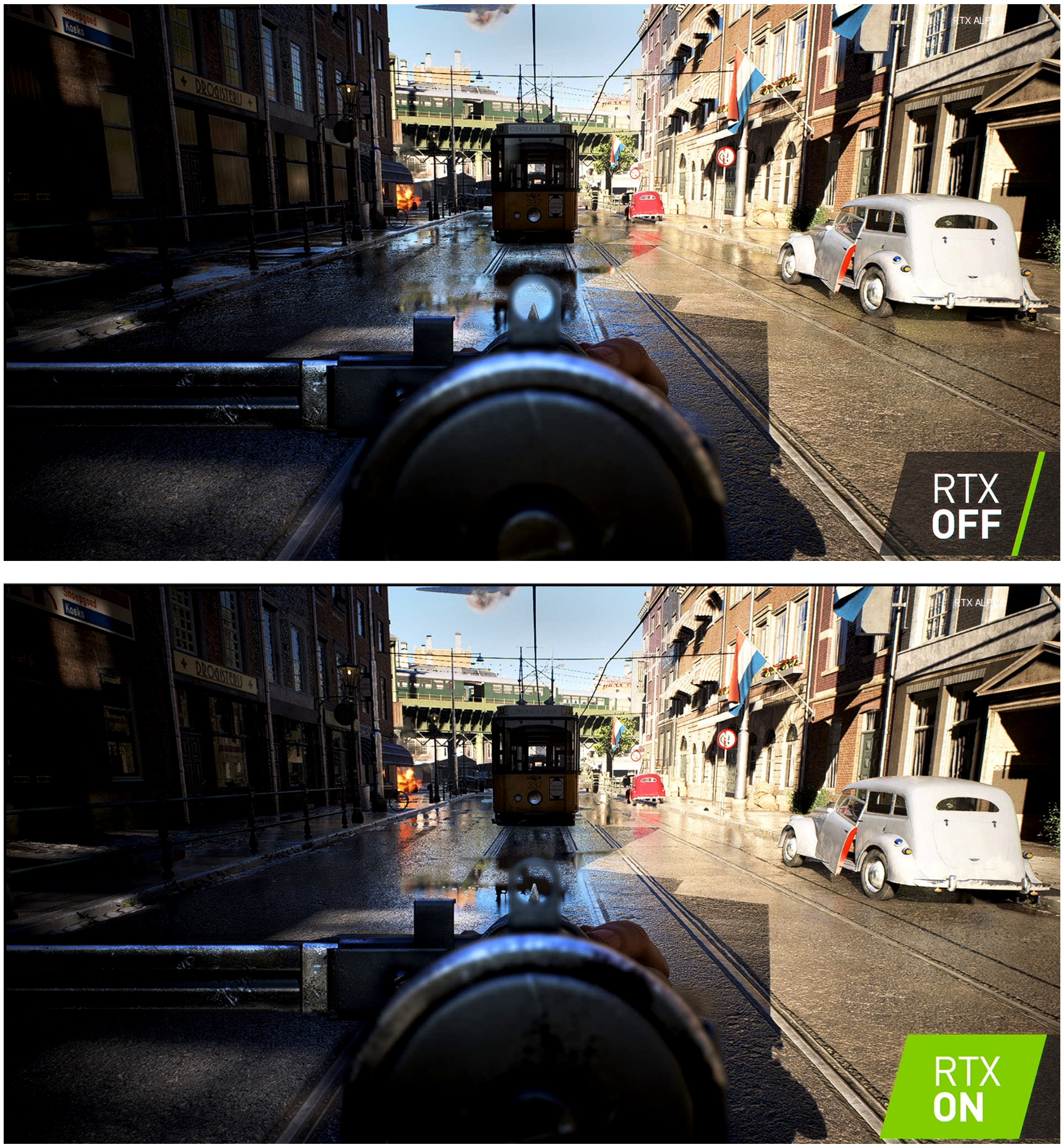
The following figures are from two upcoming game titles; *Battlefield V* and *Shadow of the Tomb Raider*. These images are using NVIDIA Turing ray tracing technology for visual effects.

- Figure 51, Scene from Battlefield V with RTX On and Off
  This scene shows another issue with non-ray-traced reflection algorithms. In this case, with RTX OFF, a reflection is partially present, but missing for the portion of the scene that is visible through the gunsight. With RTX ON, the scene looks correct.
- Figure 52, Scene #2 from Battlefield V with RTX On and Off
- Figure 53, Shadow of the Tomb Raider with RTX ON



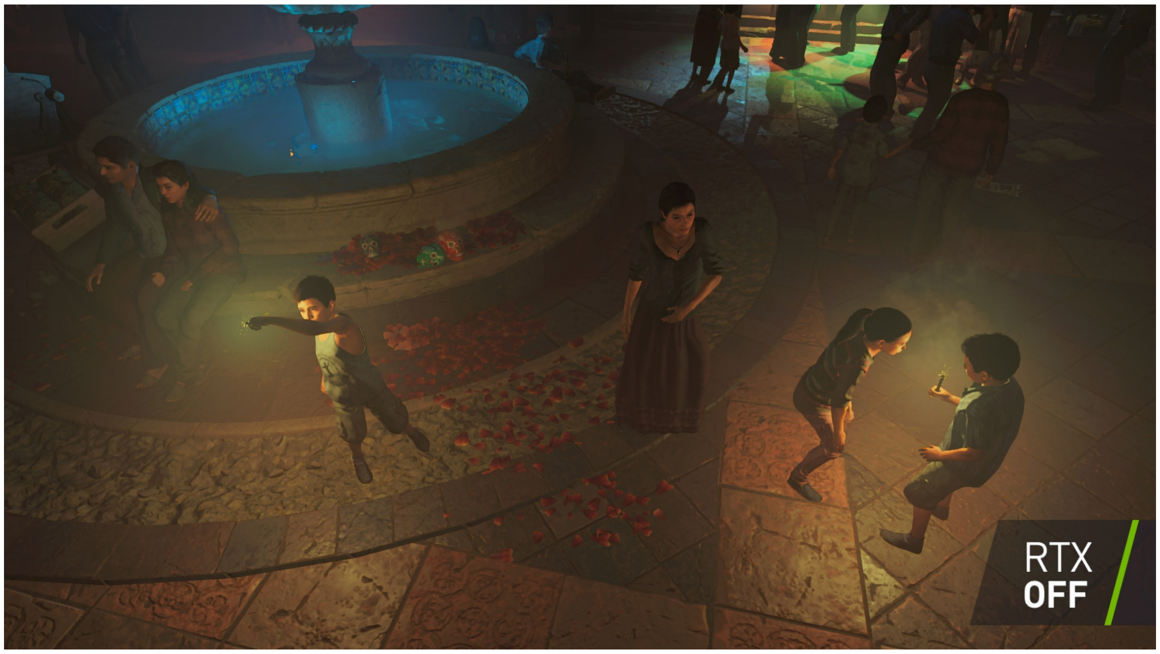
Scenes from the Alpha Version of the *Battlefield V* game from publisher Electronic Arts and developer EA. Dice is using NVIDIA RTX Technology and Turing real-time ray tracing for multiple effects in the game. You can see realistic reflections on the car from an off-screen explosion in the RTX ON scene. Such reflections are not possible with screen-space reflections without ray tracing, as in the RTX OFF scene.

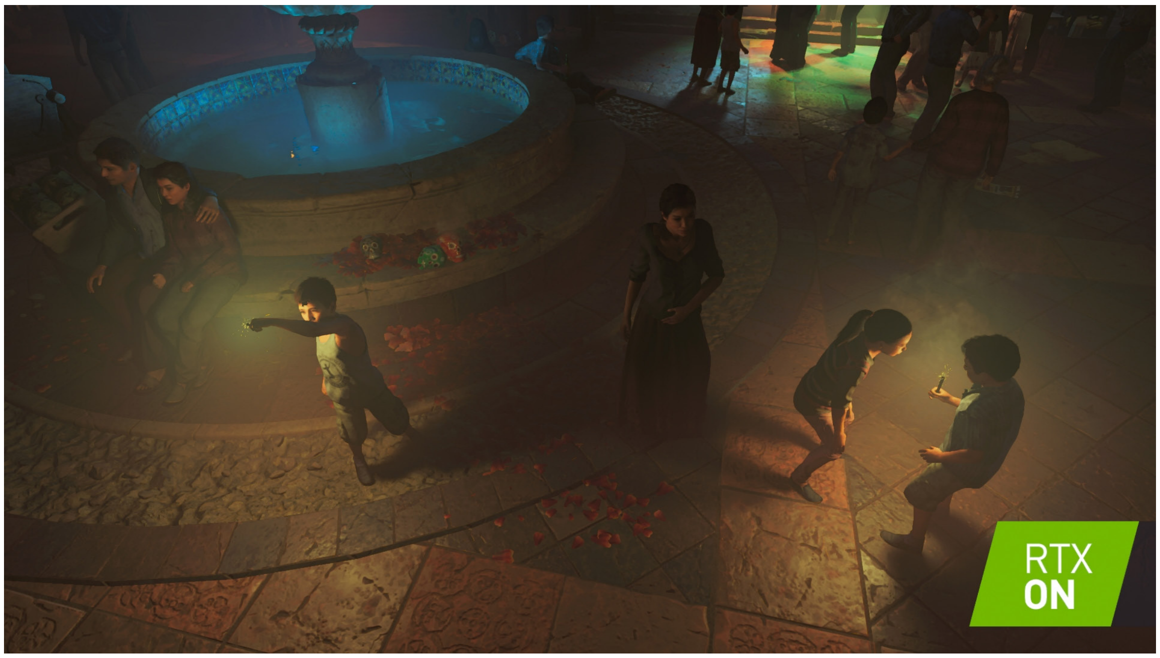
Figure 51. Scene from Battlefield V with RTX On and Off



This scene shows another issue with non-ray-traced reflection algorithms. In this case, with RTX OFF, a reflection is partially present, but missing for the portion of the scene that is visible through the gunsight. With RTX ON, the scene looks correct.

Figure 52. Scene #2 from Battlefield V with RTX On and Off





Scenes from a pre-release version of Shadow of the Tomb Raider with RTX ON vs OFF. With RTX OFF, there are no shadows cast from the sparklers being held by the children, so they look like they are floating above the surface. With RTX ON, the shadows are correct.

Figure 53. Shadow of the Tomb Raider with RTX ON

In summary, a number of technologies have all come together to enable real time ray tracing with Turing:

- Hybrid rendering
  Reduces the amount of ray tracing needed in the scene by continuing to use rasterization for the rendering steps where it is still effective, while using ray tracing for the rendering steps where rasterization struggles.
- Denoising algorithms
  Reduce the number of rays that need to be cast per pixel to produce an accurate result.
- BVH algorithm
  Used for ray triangle intersection, which makes the ray tracing operation much more efficient by reducing the number of triangles that actually have to be tested to find a hit.
- RT Cores
  All of the optimizations above helped to improve the efficiency of ray tracing, but not enough to make it close to real time. However, once the BVH algorithm became standard, the opportunity emerged to make a carefully crafted accelerator that would make this operation dramatically more efficient. RT cores are that accelerator, making our GPUs 10x faster on ray tracing and bringing ray tracing to real time graphics for the first time.