Theme Article: Hot Chips

RTX on—The NVIDIA Turing GPU

John Burgess
NVIDIA

***Abstract*—NVIDIA's latest processor family, the Turing GPU, was designed to realize a vision for next-generation graphics combining rasterization, ray tracing, and deep learning. It includes fundamental advancements in several key areas: streaming multiprocessor efficiency, a Tensor Core for accelerated AI inferencing, and an RTCore for accelerated ray tracing. With these innovations, Turing unlocks both real-time ray-tracing performance and deep-learning inference in consumer, professional, and datacenter solutions.**

At NVIDIA, we imagined a future where real-time graphics combines rasterization, ray tracing, and deep learning. NVIDIA's latest processor family, the Turing GPU, was created to realize that vision.

The largest Turing GPU, Titan RTX, comprises 18.6 billion transistors in a 12-nm process, including several thousand programmable processing elements, industry-first support for GDDR6 memory, and high-bandwidth NVLink for multi-GPU connectivity.

While Turing is packed with features and horsepower,[1] we made fundamental advancements in several key areas—streaming multiprocessor (SM) efficiency, a Tensor Core for AI inferencing, and an RTCore for ray-tracing acceleration.

Digital Object Identifier 10.1109/MM.2020.2971677
Date of publication 4 February 2020; date of current version 18 March 2020.

Efficient New SM Core

In 2017, we introduced the Volta GPU architecture, targeted at high-performance computing and deep-learning training in the V100 solution.[2] Turing builds aggressively on the foundation of Volta, bringing those advancements into the consumer GPU space.

Enhanced L1 Data Cache

Compared to the previous consumer generation, Pascal, Turing SM has twice the instruction schedulers, simplified issue logic, and leverages a large, fast, low-latency L1 data cache (Figure 1, left).

This new L1 cache architecture exploits a powerful topological change. In Pascal, global memory and texture data were accessed through the fixed-function texture processing pipeline. In Turing, global and shared memory accesses share a path directly to the RAM, providing twice the L1 bandwidth and a significant

Published by the IEEE Computer Society

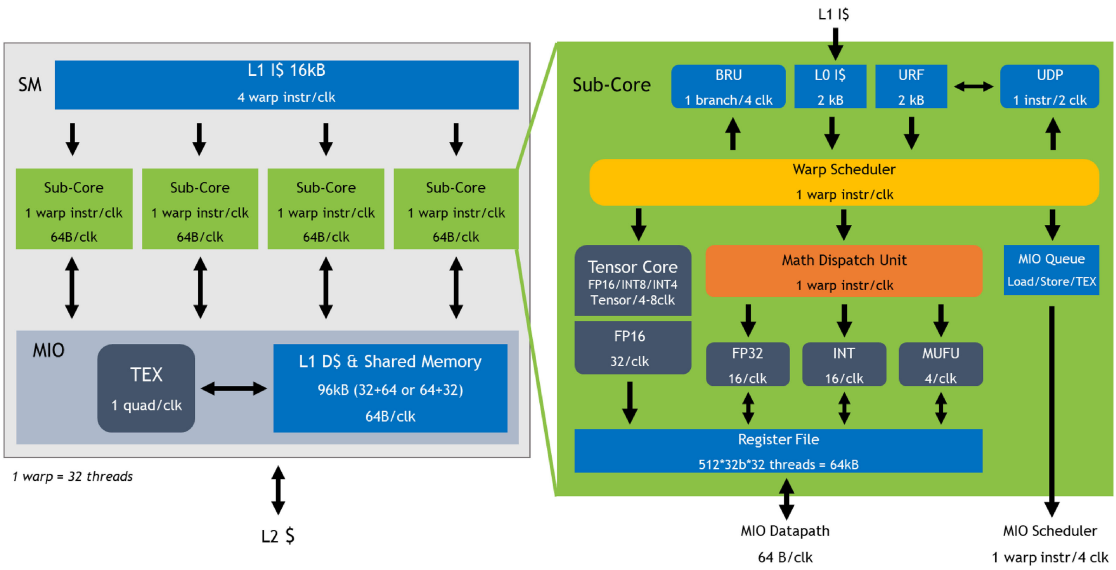

Figure 1. Turing GPU SM, comprising four subcores and a memory interface (MIO). Math throughput, memory bandwidth, L1 data cache topology, register file and cache capacity, and a new uniform datapath were all designed or modified to increase processor efficiency over the previous generation.

reduction in hit latency. Because the L1 data cache and shared memory RAM are unified, the tagged capacity can be configured based on the workloads running on the GPU.

The SM accesses an L2 cache across a shared crossbar, and on Turing, the L2 capacity doubled to 6 MB.

Concurrent Execution of Floating-Point and Integer Math

Inside each of the four subcores of the Turing SM (Figure 1, right), we doubled register file capacity, redesigned the branch unit, and added fast general purpose FP16 math.

Saturating the FP32 datapath requires only half the issue bandwidth, allowing another datapath to execute concurrently. Figure 2 shows the performance impact of concurrent execution of floating-point and integer instructions, across several gaming workloads. On average, 36 integer instructions co-execute with every 100 floating-point instructions. Typical integer operations include address computation and floating-point compares.

Uniform Datapath and Uniform Register File

The Turing SM exploits redundant computation and data across multiple threads in a warp, while preserving the single instruction multiple thread (SIMT) with independent thread scheduling model introduced in Volta.[3]

In a traditional SIMD/vector architecture (Figure 3, top), control flow resides on a scalar thread, and the developer and compiler promote certain operations to the SIMD/vector lanes to run in parallel. A vector execution mask specifies which lanes ignore the vector operation, if needed. In our solution (Figure 3, bottom), control flow resides in each SIMT thread. On Turing, when warp-uniform data are detected, the compiler with hardware assist promotes operations to an independent uniform datapath, essentially a "reverse vectorization." This promotion is valid even if one or more of the threads in the warp are currently diverged within its own control flow.

This optimization is invisible to the programmer, but the following simplified machine code snippet demonstrating bindless constant memory access illustrates the mechanism:

```
ULDC.64     UR20, [UR6 + 0x18], !UP7
UIADD       UR6, UR3, UR10
FMUL        R15, R4, cx[UR20][0x64]
ULOP.AND    UR2, UR3, 0xfffff
```

The U-prefixed, or uniform, instructions operate on U-prefixed registers, and a subsequent

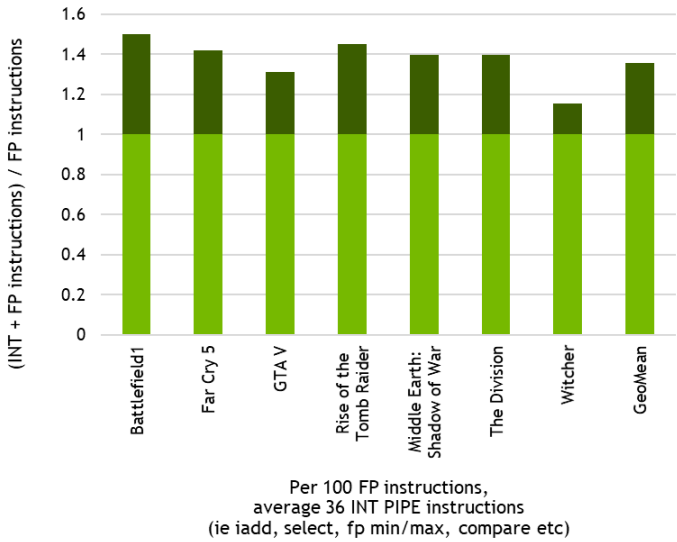

Figure 2. Concurrent execution of floating-point and integer math, across representative shaders from several popular games.

SIMT instruction uses the uniform result to index into a constant bank. This indirection avoids the synchronization overhead of dynamically updating a block of constant memory still in use by other work elements. Simply enabling bindless constants using the uniform datapath yielded a 12% speedup on Forza Motorsport 7.

With these and other efficiency improvements, SM performance increased by more than 50% across a wide variety of shaders (Figure 4). Some workloads show a benefit of over two times.

Tensor Core for Accelerated Deep Learning

Alongside the new SM efficiency improvements, we added a specialized AI accelerator called a Tensor Core. The first Tensor Core was introduced in Volta, and again, we leveraged that technology as a foundation upon which to innovate further. Integrating this new Tensor Core in the SM enables real-time deep-learning inference on a consumer GPU for the first time.

The Tensor Core efficiently accelerates the computation of matrix multiply–accumulate operations, the fundamental building block of deep learning. On Pascal, matrix multiply–accumulate operations are computed serially, while on the Turing Tensor Core this operation executes in parallel across tiles of streamed data, achieving a throughput of 114 TFLOPS of FP16 math, or 228 TOPS 8-bit integer math, or 455 TOPS 4-bit integer math, on a GeForce GTX 2080Ti GPU.

The Tensor Core is tightly integrated into the SM subcore (Figure 1). It performs a single multithread collaborative matrix math operation over four to eight clocks, which saves operand and memory bandwidth by transparently sharing data across threads.

This fine-grained integration provides maximum algorithmic flexibility. For example, different activation functions or batch normalization variants can be interleaved on the general purpose datapaths alongside the matrix operations executed on the Tensor Core. This integration also naturally leverages the large capacity and bandwidth of the register file and shared memory.

As a result, the Tesla T4 datacenter product accelerates all AI workloads flexibly, achieving speedups of over five times the Pascal-based Tesla P4 solution on DeepSpeech2, and up to 36x that of CPU-based solutions for natural language processing (Figure 5). The capability to accelerate all AI workloads is critical, as multiple networks chained together provide higher level solutions, e.g., speech recognition coupled with natural language processing to enable interactive agents.

We expect deep learning to disrupt gaming and professional graphics as it has disrupted other application spaces. Examples of dynamic neural graphics include deep learning supersampling (DLSS), which both anti-aliases and upscales game images for dramatically improved frame rates and image quality; style transfer and content creation with GauGAN,[4] which enables an artist to paint a simple segmentation map image and automatically generate beautiful and plausible landscape paintings; and AI slow-motion video on a professional workstation, automatically generating the missing video frames from a standard video. Deep-learning-based graphics has only just begun, and Turing makes it possible in real time.

RTcore for Accelerated Ray Tracing

We also invented a new accelerator for ray tracing, making Turing RTX the first shipping GPU with hardware-accelerated ray tracing. The RTCore provides up to 10 GRays/s traversal and intersection throughput. Overall, Turing RTX is seven

times the Pascal ray-tracing performance, meaning that real-time ray tracing is finally available.

Figure 6 is a screenshot from the interactive demo Attack from Outer Space, built with Epic Games' Unreal Engine 4 and a 2019 DXR Spotlight award winner.[5] The image demonstrates multiple advanced rendering techniques including soft shadows underneath the cars on the street, and glossy reflections in the robot's chest plate and the puddles on the ground. These are calculated dynamically in real time in a destructible, animated environment, and made possible by accelerated ray tracing.

Beyond using rays for shadows or reflections, one can employ even more advanced techniques to generate ever more realistic images, e.g., path-traced global illumination, in which rays represent virtual photons in a physical simulation. Figure 7 is a simplified diagram representing the path-tracing process.

A ray, represented as an origin and a direction, is used to test what geometry is visible from a point in the scene.

- A camera ray, labeled 1 in the figure, tests what geometry is behind each pixel in the image being rendered. For performance, a hybrid renderer may use traditional rasterization for this step.
- A reflection ray, labeled 2A, tests what light scattered toward the viewer from some other part of the scene. For rougher reflections, one direction is randomly chosen from a distribution of likely scattering directions. Refraction rays, labeled 2B, may also be appropriate for transparent materials.

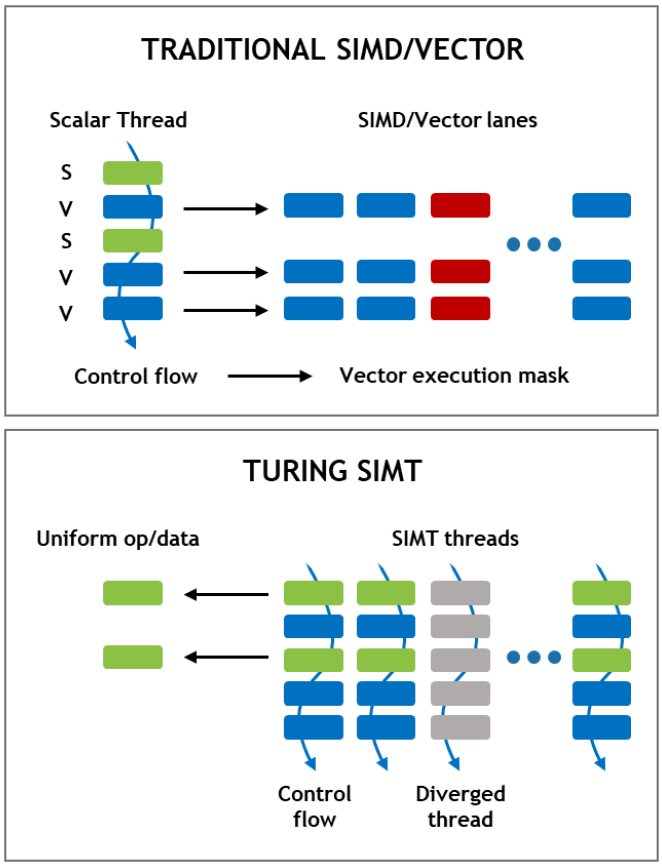

Figure 3. Traditional scalar/vector architecture (top) versus Turing SIMT with uniform datapath (bottom). SIMT avoids the requirement for programmers to explicitly vectorize parallel tasks. The promotion of uniform operations and data for efficiency works even in the presence of diverged threads in an SIMT warp.

- A shadow ray, labeled 3, tests whether an occluder blocks light from arriving at a surface. Rather than test all light sources, typically only one is randomly chosen, but a special case for the sun, labeled 4, may be desired.

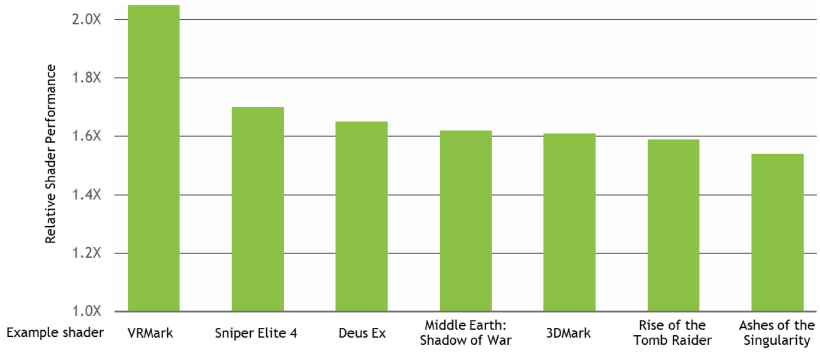

Figure 4. Representative shader workloads extracted from various games and graphics benchmarks show up to two times performance benefit from the Turing SM.

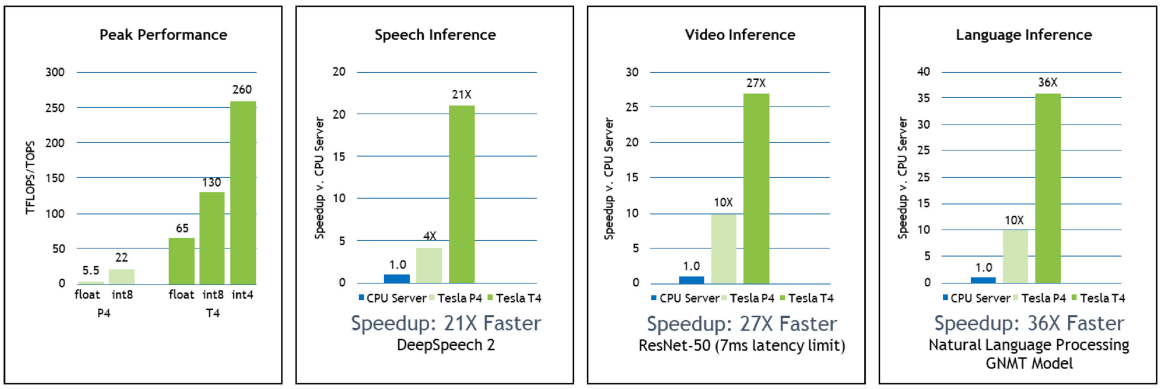

Figure 5. Accelerated deep-learning inference with Turing-based Tesla T4 datacenter solution, with speeds up to 36 times faster than CPU-based servers. T4 can flexibly apply chained deep-learning tasks like speech recognition followed by natural language processing to enable higher level solutions like intelligent agents.

- These rays are generated recursively (labeled 5, 6, and 7).

Chains of rays representing reflections, refractions, and light source queries create paths between the camera and the light sources to determine the color of light transported to each pixel in the image. This technique is commonly used to generate CGI movies; however, due in part to the inherent random sampling, the brute force computation typically takes many CPU hours to generate a single frame. Recent GPUs reduce the cost to minutes or seconds, but real-time performance has been unachievable.

The fundamental building blocks of the algorithm are as follows:

- sampling (What direction to shoot a ray?);
- traversal and intersection (What did the ray hit?);
- material evaluation (How does the light scatter at that hit point?).

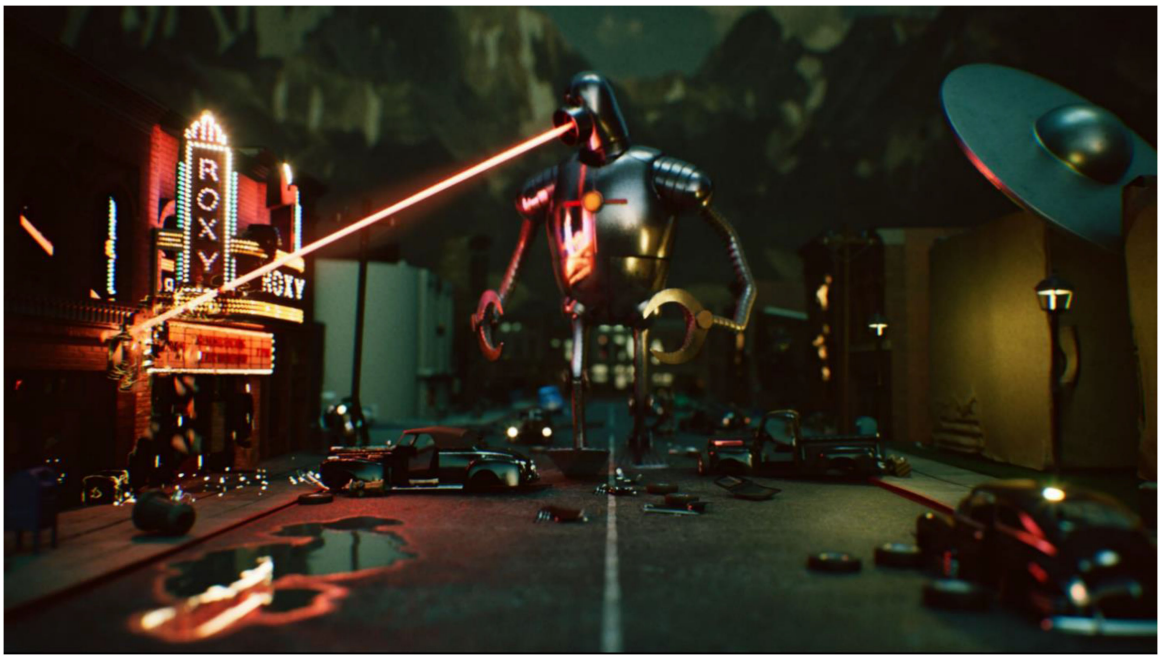

Figure 6. Attack from Outer Space interactive ray-tracing demo by Christian Hecht. Soft shadows, glossy reflections, and indirect illumination combine to produce realistic lighting in the UE4 game engine.

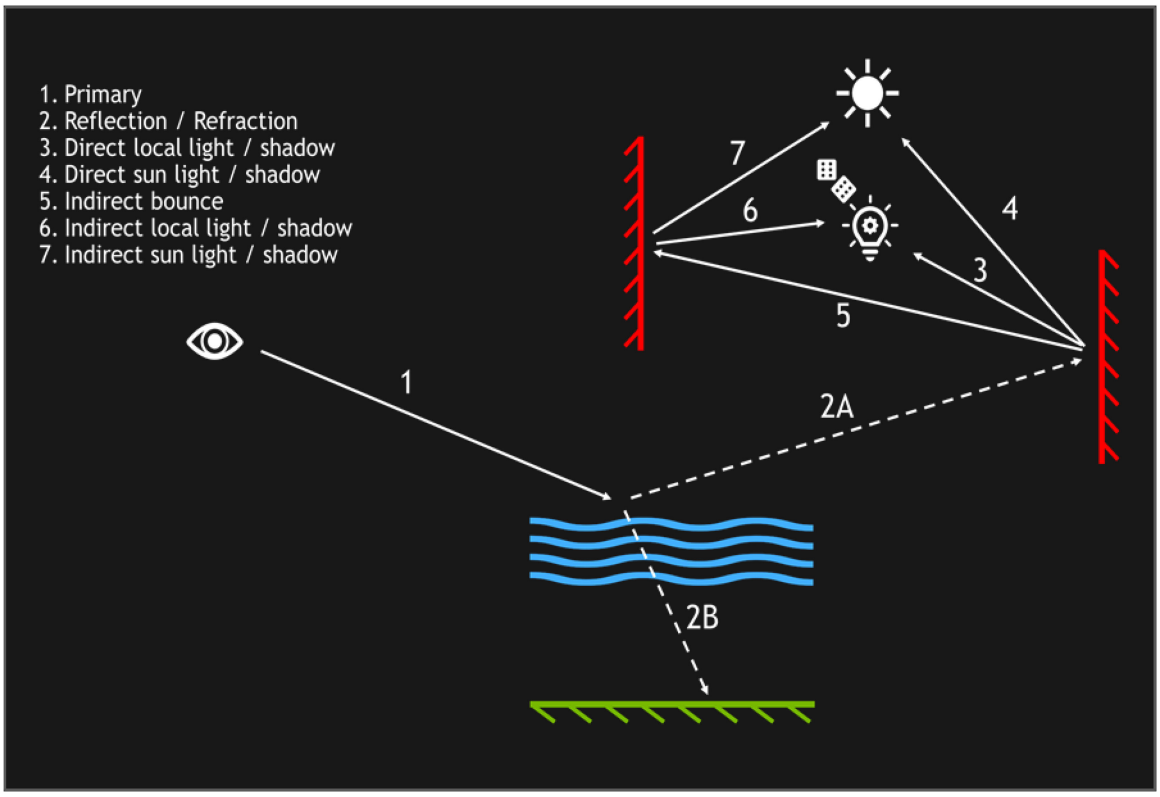

Figure 7. Path-traced global illumination, simplified. Primary/camera rays, reflection/refraction rays, and shadow rays combine recursively to simulate the color of light scattered from the scene to each pixel in the rendered image.

Sampling and material evaluation are not yet suitable for fixed-function acceleration. Techniques for these differ across renderers, requiring significant flexibility. Traversal and intersection, however, are the most expensive components of global illumination, has essentially annealed to a common algorithm in the industry, and is ripe for further acceleration.

In pre-RTX GPU ray tracing (Figure 8, top), traversal and intersection were done on the SM core. To avoid testing every triangle in a scene against each ray query, a bounding volume hierarchy or a tree of axis-aligned bounding boxes is created over the triangles in the scene.

When a ray probe is launched, this tree is traversed, and each successive child box is tested against the ray, culling the distant geometry efficiently. At the leaves of the tree, the actual triangles making up the surfaces in the scene are tested against the ray. This tree traversal is fundamentally pointer-chasing through memory interleaved with complex, precision-sensitive intersection tests, taking thousands of instruction slots, and potentially tens of thousands of cycles of latency per ray. Finally, when the appropriate hit point is located for the ray, the material at that point is evaluated.

On Turing RTX, the new RTCore replaces that software emulation (Figure 8, bottom), performing the tree traversal and the ray/box and ray/triangle intersection tests. A ray query is sent from the SM to the RTCore, the RTCore fetches and decodes memory representing part of the bounding volume tree, and uses dedicated evaluators to test each ray against the box or, at the leaves of the tree, the triangles that make up the scene. It does this repeatedly, optionally keeping track of the closest intersection found. When the appropriate intersection point is determined, the result is returned to the SM for further processing.

The RTCore is faster and more efficient than software emulation and frees up the SM to do other work including programmable sampling or material shading in parallel. We carefully interfaced the RTCore and the SM for both

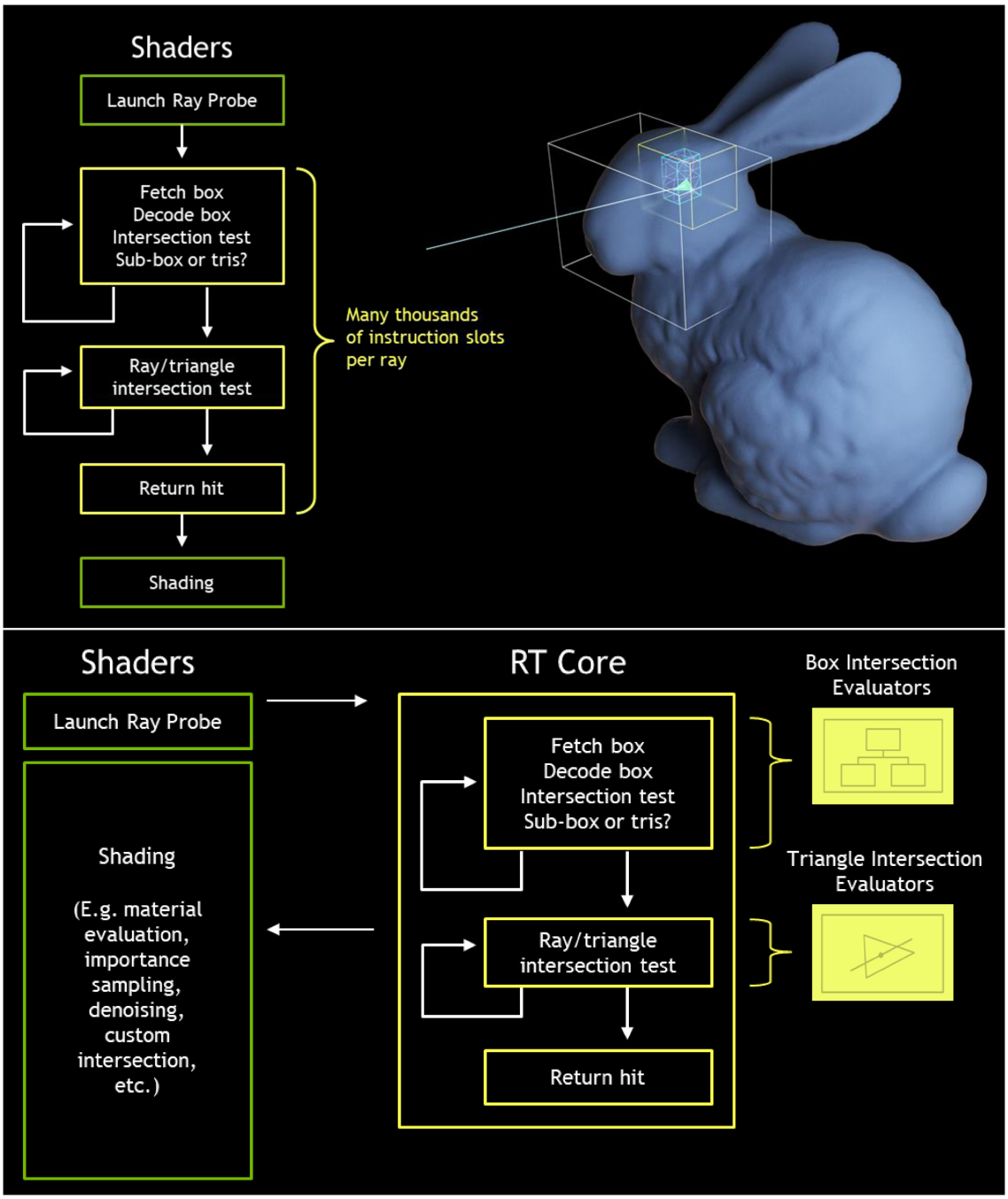

Figure 8. Ray tracing before (above) and after (below) Turing RTX. The RTCore performs ray/scene traversal and intersection tasks, freeing the SM to do concurrent work such as material evaluation and denoising.

performance and flexibility, enabling a developer to optionally create custom intersection programs that run on the SM for non-triangle geometry such as spheres that traditional rasterization cannot easily handle.

Figure 9 shows the performance of one frame of Quake II RTX, a game recently remastered to use path tracing. Each plot is a timeline of the frame on a different configuration. At the top, on Pascal, path tracing is possible, but five frames per second (fps) is much too slow to be playable. In the middle, on Turing without the RTCores enabled, the more efficient design is already twice as fast at 10 fps. The purple and gray on this

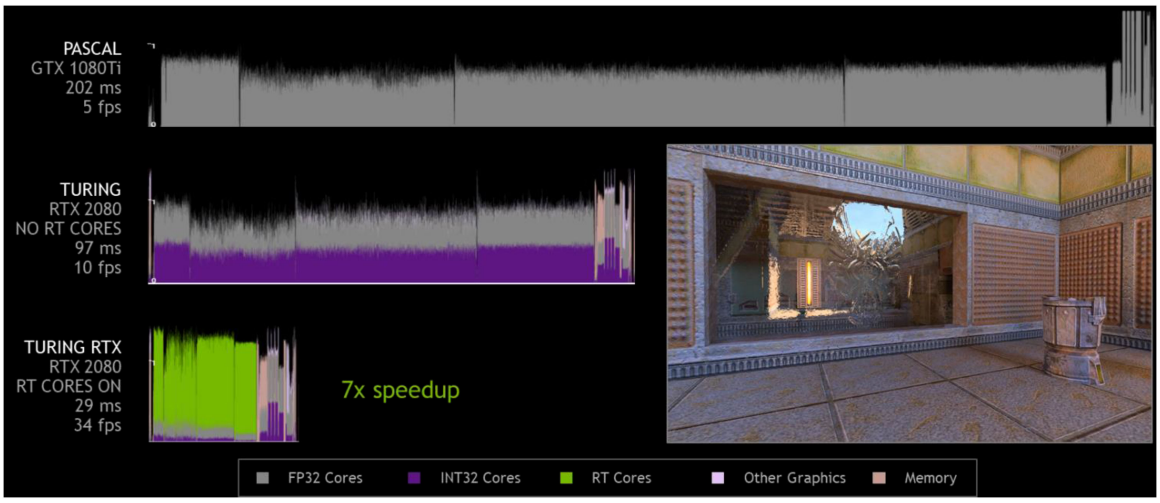

Figure 9. Path-tracing performance on one frame of Quake II RTX. Turing efficiency improvements yield a two-times speedup over Pascal, while Turing RTCores provide a further speedup for a total seven times faster frame rate and real-time performance.

timeline show the floating-point and integer datapaths executing in parallel. Enabling the RTCores, their work shown in green in the bottom plot, yields a further leap to real-time speeds at 34 fps and an overall seven times speedup.

In the rendered image of this frame (Figure 9, inset), reflections, refractions, soft shadows, indirect lighting, and more are evident, bringing a dramatically updated look to a fun game from the past.

Developers are adding ray-traced effects to upcoming games at a rapid pace as well. Every graphics API and major engine has added support. Real-time ray tracing has finally arrived.

Figure 10. Professional rendering before (left) and after (right) AI-based denoising, using a Quadro RTX workstation. The SM and RTCores are used to trace a few (noisy) paths per pixel, and the Tensor Core accelerated AI denoiser estimates the completed image, together providing a speedup of several orders of magnitude versus a CPU server.

Turing: Greater Than the Sum of Its Parts

Because these three key architectural investments were carefully integrated, they make Turing far greater than the sum of its parts. By leveraging the SM, RTCore, and Tensor Core simultaneously, RTX-enabled professional film and design renderers can now achieve interactive path tracing with high-quality materials and AI denoising.

At NVIDIA, we envisioned the future of graphics, but it requires greater efficiency and new breakthrough acceleration. With a new more efficient SM that is more than 1.5 times faster than the previous generation, new Tensor Cores, which unlock real-time deep-learning inference, and the new RTCore, which yields more than 7 times faster ray-tracing performance, the Turing GPU is built to realize that vision for next-generation graphics.

Figure 10 shows a path-traced image in split screen. On the left, a few randomized paths per pixel have been ray traced and the materials evaluated. The image is noisy due to the random sampling inherent in the path tracing algorithm. Noise-free images often take thousands of samples per pixel to converge, which is slow even with accelerated ray tracing.

However, an AI-based denoiser[6] using the Tensor Cores can estimate the completed image with as few as 1% of the typical samples, efficiently removing the sampling noise in a post-process pass (Figure 10, right). The SM, RTCore, and Tensor Core work together efficiently to render the final image. This level of quality and interactive performance in a professional rendering would not be possible with any of these three elements missing.

CONCLUSION

At NVIDIA, we envisioned the future of graphics, but it requires greater efficiency and new breakthrough acceleration. With a new more efficient SM that is more than 1.5 times faster than the previous generation, new Tensor Cores, which unlock real-time deep-learning inference, and the new RTCore, which yields more than 7 times faster ray-tracing performance, the Turing GPU is built to realize that vision for next-generation graphics.

REFERENCES

1. NVIDIA, "NVIDIA Turing GPU architecture: Graphics reinvented," 2018. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf
2. NVIDIA, "NVIDIA Tesla V100 GPU," 2017. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf
3. J. Choquette, O. Giroux and D. Foley, "Volta: Performance and programmability," *IEEE Micro*, vol. 38, no. 2, pp. 42–52, Mar. 2018.
4. T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu, "Semantic image synthesis with spatially-adaptive normalization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2337–2346.
5. C. Hecht, "NVIDIA DXR spotlight winners—Attack from outer space," 2019. [Online]. Available: https://developer.nvidia.com/dxr-spotlight-winners
6. C. R. A. Chaitanya *et al.*, "Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder," *ACM Trans. Graph.*, vol. 36, no. 4, Jul. 2017.

John Burgess is a Senior Director of GPU architecture with NVIDIA, Santa Clara, CA, USA. Most recently, he has led the development of the SM architecture for the Volta and Turing GPU families, including dedicated hardware for accelerated deep learning and ray tracing. He received the Ph.D. degree in physics from the University of Texas at Austin, Austin, TX, USA. Contact him at jburgess@NVIDIA.com.