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).

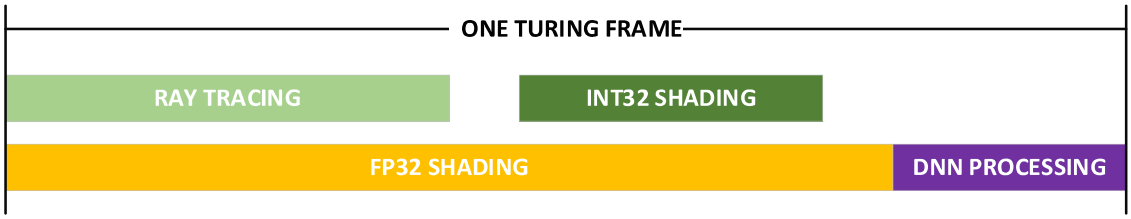

Figure 43. Workload Distribution Over One Turing Frame Time

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%.

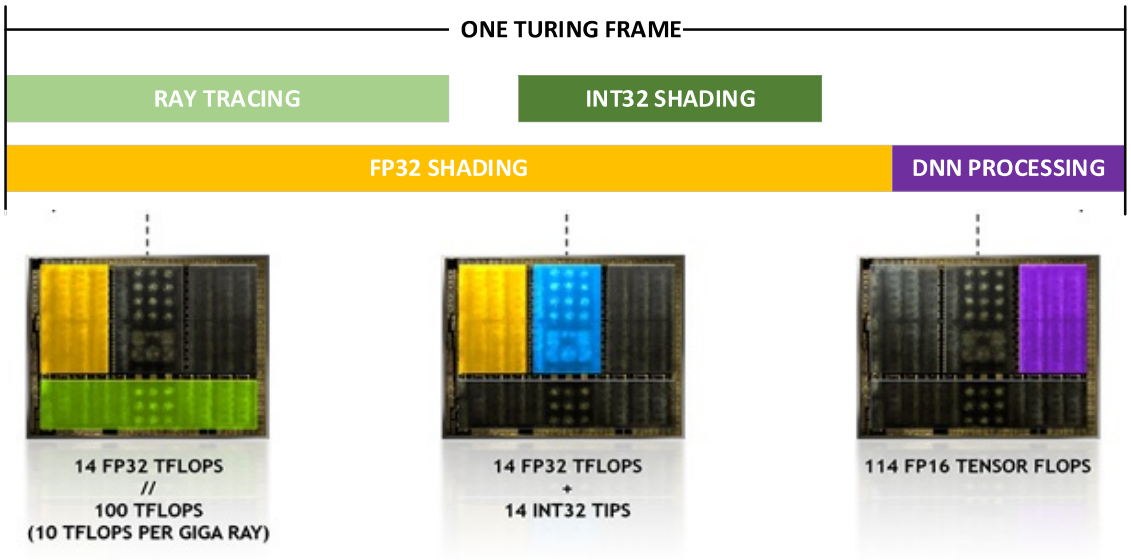

Figure 44. Peak Operations of Each Type Base for RTX 2080 Ti