Exploring The Use of SIMD and SIMT Devices in Capturing Real-Time Photorealism

Alan Pfeil, Camille Vo

Sponsor: PhD. Nuno Sena from the University of California, Merced

Table of Contents

Symposium/Special Report

Introduction:

"Ray tracing is not slow - computers are"[2-7] . Computers can perform basic calculations in mere nanoseconds, however, when many calculations must be performed upon large quantities of data, sequential nanoseconds add up considerably to where functions can take days to execute. As the demands of big data get larger and we approach the limits of silicon's efficiency, parallel computing is the relatively uncharted territory to dramatically increase the speed of these intense computations. CPU performs a single calculation on a single data SISD[2-5] point at a time, and when that one is complete, the next calculation begins. Often times in a program, the same calculations will need to be done over and over with many different data points. What if we could perform all of these calculations at once using more complex computer hardware?

The first SIMD[2-5] device was developed in 1966 (Illiac IV). SIMD devices today can be seen most often in GPUs, where each pixel is treated as a data point and a display function is applied to every data point within that vector. These devices are able to perform a single calculation on many pieces of data simultaneously, so they can be useful for repetitive calculations. Traditionally, a program runs on a SISD CPU unless the program specifies it wants to use the GPU for parallel computations (mostly to display something). However, if we were to broaden use for SIMD devices, we could solve problems in computer science related to taxing algorithms through vectorization. In our case, we look at the ray tracing algorithm to achieve photorealism and explore Nvidia's real-time solution (RTX) using SIMD (technically SIMT) technology to show both the relevance/exploitative quality of vectorization and the reason why, in Nvidia's case, it's hard to bring these innovations out into general use.

Concern:

Concerns with the state of real-time photorealism is present within the perspective of the consumer, the programmer, and the creators. This concern is a spiraling/cyclical one, so the solution to the concern is especially tricky and complex for it requires one or two of the groups to take action in order to leave the spiral. Like many scientific innovations/researches, the problems lie within different scopes of interest. Commonly, scientific hurdles are a result of internal (scientific, mathematical, and/or from lack-of-breakthrough) forces *and* external (user trends, investor demands, governmental interest, social welfare interests, contractual goals met/limited by) forces. It's what computer scientists refer to as "deadlock": when one resource is waiting on another, and the other resource is waiting on the next, and so forth until we have a circular queue of indefinite waiting where one of the threads needs to take the first step out of the circle.

Critical Overview of Chapter 1:

In Chapter 1, Vo explores the use of SIMD devices in research computation and the development and use of SIMT. Computers typically perform calculations sequentially, meaning that a single calculation is executed on a single piece of data and once it is finished, the next calculation begins. Parallel computing utilizes hardware to perform multiple calculations simultaneously and is different from vector computing, where a single instruction is performed on many pieces of data, a vector, simultaneously. The type of hardware that executes this kind of programming is called SIMD, Single Instruction Multiple Data, and is most commonly seen in GPUs, Graphics Processing Units. A GPU treats each pixel of the screen as an individual data point, and therefore can generate images very quickly because the calculations required are able to be performed for each pixel simultaneously. This… <Camille - your chpt->

Critical Overview of Chapter 2:

Real-time photorealism is a hot topic in the computer industry- especially the gaming industry- since as early as the late sixties in the first annual ACM SIGGRAPH computer graphics and interactive techniques expo[2-7]. Within the expo of '86, an equation was derived to explain the natural characteristics of rendering on a computer; the amount of processing power a computer must run through to output a result that is final. Fast forward a couple of decades and we've learned to vectorize the rendering process as a whole. Within the consumer's scope and point-of-view, this vectorization comes in the form of graphics cards (GPUs) used to apply effects in a 3D environment to eventually be rasterized and displayed on your screen[2-3]. For an animation studio this form of vectorization comes in the form of supercomputers in order to further speed up the rendering equation's frequented use. One implication of the equation within the computer (formally non existent in real-time) real-time graphics realm is the method of *ray tracing*. Whether for modelling, movies, architecture, or gaming, this "hybridized" method of rasterization (although technically separate from it) is very time consuming hence its iterative use for all light rays in the environment. This is however just one step towards achieving real-time photorealism; it's not the sole step -nor an exclusive step- towards real-time photorealism either. This does not include all of the complexities of external pressures and desires from a consumer and scientific standpoint, which are topics of discussion within chatper 3.

Critical Overview of Chapter 3:

Due to issues with programmer motivation, widespread distribution, and a lack of focused consumer demand, it is hard to get the full implementation of scientific innovations out to the public. With real-time ray tracing, consumers were looking for Nvidia's newest flagship lineup of GPUs to have an expected performance increase over previous generations, but for the price a justifiable performance gain was nowhere to be seen in place of where real-time tracing was implemented. With Programmers, real-time ray tracing was a big overhaul towards new technology that most people aren't ready for yet. Since it's unfamiliar to the average programmer, sticking to other robust rendering methods is seen to alleviate a future headache when programming. Thus, early adoptions of real-time ray tracing in video games are executed via contractual obligation by Nvidia upon willing developers. Only recently has ray tracing been added to Unreal Engine 4.2, so its impact/usage with future games has yet to be known. Part of the time, consumers don't know what they exactly want when it comes to the stepping stones before a demand is met. In other words, if we are interested in true real-world simulations we would first need the support of real-time photorealistic graphics in a traditional environment before we could even attempt real-time photorealism in an augmented/virtual environment (AR/VR). It was once stated by Steve Jobs that looking back and connecting the dots backwards from a staple innovation is easy, but predicting and imagining these dots before-hand is incredibly hard; it takes both skill and creativity and requires much speculation and trial before a forward dot could be drawn.

Significance:

Currently, in the field of computer science and engineering, we are pushing the boundaries of innovation both due to the consumer's rapidly-growing interest in computing technology and the vast support from motivated (optimistic) investors. These boundaries we're pushing include self-learning AI, augmented reality (holographics), virtual reality, big data, and real-time photorealism. These boundaries all have one thing in common: they require supporting hardware to reach what they set out to achieve. Self-learning AI utilizes SIMD architecture to browse the web and reach a thorough (and unique) answer to a question, augmented and virtual reality both require photorealistic graphics support (in their respective environments) to bridge the gap between virtual and real, and big data algorithms include repetitive use of the same functions to crunch big numbers which requires the use of vectored devices. Akin to the method in calculating big pieces of data at once in a vector (big data and self-learning AI) is the strategy used for real-time photorealism with ray tracing (3D applications including virtual/augmented reality). Before the latter, we must acknowledge the necessary and intermediate steps so that we can move towards the final goal; in this case, we're still on the step towards achieving real-time photorealism- but this has an impact for the other topics as well hence the method of alleviating the rendering process is close to that of calculating big data and searching through numerous pages: this all revolves around SIMD Flynn taxonomy for computer architecture. With this architecture we can supply the appropriate hardware for each problem so that the runtimes of the programs are adequate. Since each boundary includes the "rendering" or "crunching" of large systems/pieces of data, we must implement hardware that follows this dynamic.

PAGE BREAK INTENTIONAL

Individual Research SECTIONS of Special Report/TEAM document).

INSERT YOUR PAPERS IN PROPER ORDER.
Create a Chapter Heading and information lines .

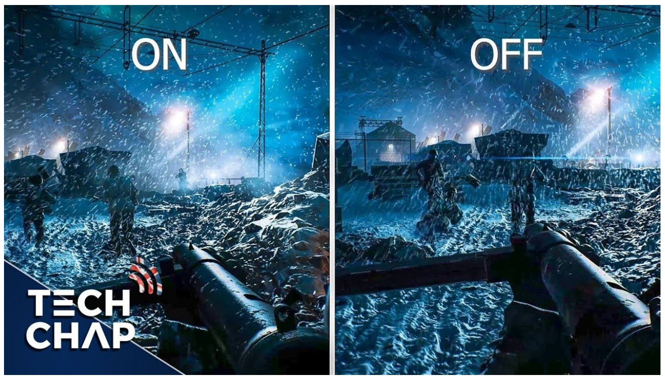
Chapter 1
TITLE

Vo, Camille

Introduction

Despite past decades of research that has gone in to making real-time ray tracing a reality, Nvidia is having a difficult time getting consumers to get on board with their new graphics cards. It can be frustrating on the end of the researcher to see that years of development mean nothing in the eyes of a consumer. The current predicament is that since ray tracing is a new technology in the world of gaming, there isn't much support for developers who want to utilize it by switching to vector programming. Since it's difficult for developers to make games, consumers have no reason to invest in Nvidia's new card because there's not enough games that utilize it to justify the purchase. Since there is no consumer interest, there's not much incentive for programmers to learn a new style of programming for a game that won't sell. And the cycle continues. Nvidia's current solution is to provide financial incentive for programmers to utilize ray tracing, which has potential to break the cycle. However, this fight for consumer interest is not only unsurprising, it's typical. Nearly all tech startups and new technologies fail commercially, and the more out of the ordinary, the smaller the chance of success (cite). Consumer unfamiliarity and distrust play a large role when it comes to their adoption of new technology (Ziamou). A study conducted by Baruch College in 2002 proved that when consumers were presented with a product with an unfamiliar interface and unfamiliar functionality, information about the interface drove them away, whereas less product information let them focus on the functionality without worrying about what wouldn't work. When consumers were presented with a product with an unfamiliar interface but familiar functionality, the reverse was true. Consumers preferred the product more when they were given more information about the technology itself, because the extra information puts the focus on the new technology itself and can quell any fears about it not working as expected. However, Nvidia's ray tracing enabled GPU's could be considered familiar interface, because they work as any other graphics card, but with new functionality because the can perform real time ray tracing. By extrapolation, more information given to consumers on the interface will be detrimental because not only will it distract from the functionality itself, it will give them more ideas as to how the product could potentially fail.

Since real-time ray tracing is brand new, consumers are also unable to picture what it would be like to have it. For many, the graphics they're used to are "good enough" and do not interfere with gameplay. They're likely used to being promised game-changing graphics, then finding only marginal improvements after investing. It's easier for this gamer to visualize ray-tracing as another marginal improvement not worth investing in than to visualize themselves playing photorealistic games in real time.



Researchers can also be out of touch with the market, and often have little interest in the commerciability of their findings. After watching a short demo of the game Battlefield V with and without ray tracing enabled, it became clear that at the stage it is now, only a graphics enthusiast would notice enough of a difference to make the splurge. However, Nvidia's developments have a huge impact in the world of science, even if they have not yet shaken the world of gaming.

SIMD (Single Instruction Multiple Data) devices are capable of performing the same operation on many data points. The most common type of SIMD devices are GPU's (Graphics Processing Units), which treat each pixel like a data point to generate the full image. The have to be quick and efficient in order to generate images quickly enough to look like real time (roughly 60 frames, or images, per second). This section discusses the pros and cons of using GPU's for an atypical use in research computations.

The two main benefits of using a parallel processing GPU are speed and accessibility. Because of their SIMD structure, many data points can be calculated at once which enables intense algorithms such as Monte Carlo methods and DNA sequencing to be completed significantly faster than if each point was calculated sequentially. However, there are limitations. There is significantly less memory space on a GPU and memory access takes considerably longer. Using global memory from the CPU takes even longer because the GPU is a seperate piece of hardware. This means that parallel computing is only efficient for algorithms that have a high ratio of computations to memory access functions. Breaking down the work down between processors and trying to vectorize data that is not naturally found in a vector also takes time. There is also a significant amount of human labor that goes in to adapting equations for parallel computing on a GPU. Therefore, this method is most beneficial to computation-heavy algorithms with few variables that are executed repeatedly over large quantities of data. Although there are limitations to using GPU's for computation, the effort put in is worth it for many researchers that do not have the access to expensive supercomputers found at large laboratories. GPU's are inexpensive, modular, and can be put into many commercial computers. They can be transported with ease and will function the same even in different devices. GPU's are also relatively easier to code because they use widely-used languages like Nvidia's CUDA or Intel's OpenCl. Therefore, the development of GPU's for research could have a huge impact be enabling researchers without access to supercomputers to perform the same intense computations they need at a reasonable rate.

Nvidia was the first to develop SIMT (single instruction multiple threads), which is comparable to SIMD but significantly more advanced. What exactly a thread is is non-trivial, but can be thought of as a short executable piece of code that completes a task. In SIMT, many of these threads can be executed at the same time, which makes it easy for a multi-part algorithm such as ray tracing to be naturally broken down into steps. In the case of PaPaRa optimization, Nvidia's SIMT enabled GPU performed nearly ten times faster than Intel's SIMD enabled GPU with the same amount of time put in to optimization for each. Memory space and memory access is still suboptimal with SIMT, but Nvidia is focusing on solving this problem by modifying MPI communications, which has shown potential in decreasing the amount of time memory access takes when using these modular devices.

To conclude, ray tracing is just the beginning when it comes to utilizing SIMD devices for intense computations. Programs that would've taken days to compute can be done in hours or less if they are properly vectorized. SIMT shows great potential, but more research must be done in order to better understand these devices and how best to utilize them. However, the time spent adapting equations for parallel computing seems to be worthwhile for science and big data, and if the trend continues we could potentially see GPUs as a mainstay in laboratory research and more advanced computing power available to simple consumers.

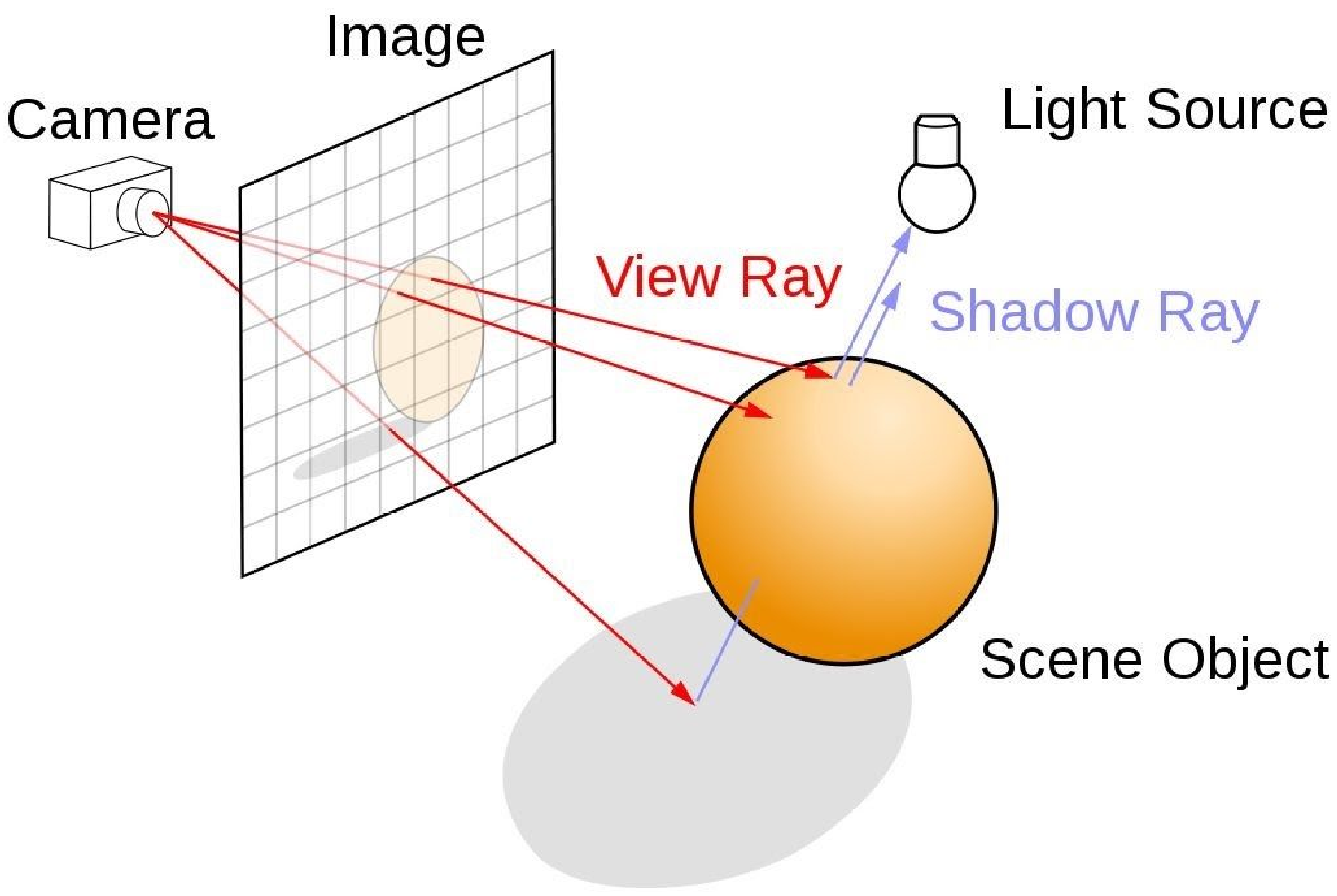
References

Only references info goes here (Put all annotations in the Appendix) …

Chapter 2
Capturing Real-Time Photorealism with Ray Tracing

Pfeil, Alan R.

Capturing Real Time Photorealism: An Annotated Bibliography
Alan Pfeil. April 9, 2019



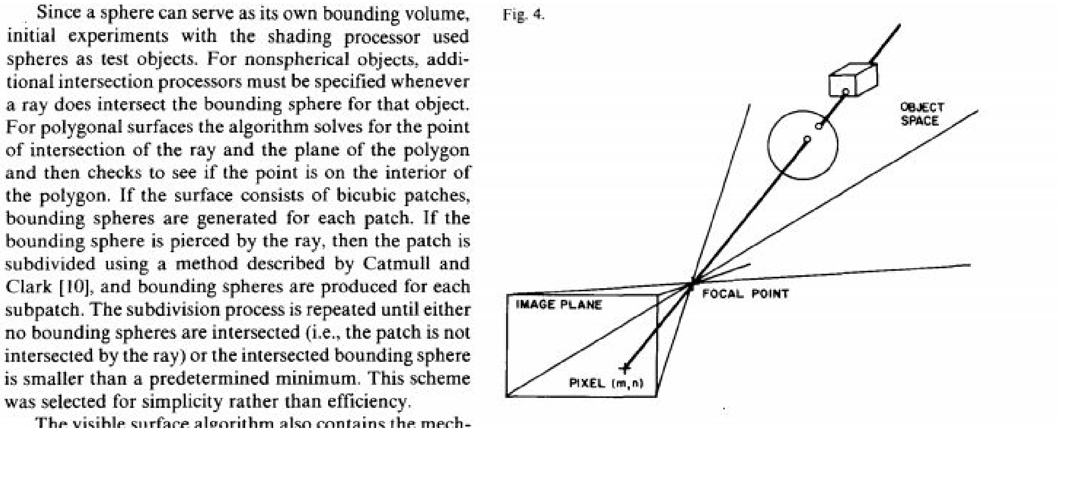
Introduction

It is a common scientific circumstance for priorly-theorized and patented technological innovations to go unsupported for various amounts of reasons that paradoxically rely on each other. In the case of computer graphics and vision, across its explicit fields, this circumstance is of no absence, although the direction and effect is significant to the industry. Within the realm of video game graphical fidelity, ray tracing is a hot topic to produce realistic three-dimensional scenes by bouncing light rays off of surfaces and models in real-time. What was previously used mostly by producer level theatrical post effects programs was made possible in the world of real time applications hence ray tracing was a process theorized and implemented as early as 1992[2-1], 1989[2-2], or even 1969[2-5]; the problem was that tracing light rays off multiple surfaces could only be done in long, tedious rendering times of which could last up to *days* of computer calculation. In 2018, Nvidia had their solution to the problem manufactured with their new flagship release of RTX cards (ray-tracing graphics cards). However, due to numerous reasons including: exclusive contractually-obligated support, unfamiliar/uninspired real-time application development, and a lack of vector-adept computer programmers, consumers are not picking up on the newest technology that computer science has to provide, leaving software support nothing to be desired by modern triple-A developers due to safety reasons for investors, leaving innovation to the smaller independent teams. This creates a spiraling effect on the technology's implementation and use; the market demand is low for this new technology due to the lack of software support, but the lack of software support is due to the low demand/necessity of the technology being produced which lags the widespread innovative process as a whole.

Assessment: The Development of Vector Hardware and The Innovative Process as a Whole

Previously, in the medium of computer runtime applications, such as video games, ray tracing was previously reserved to pre-rendered scenes and environments due to the lengthy and performance-taxing nature of the tracing algorithm's constant use. Static single-frame results usually takes up to hours[2-2] at a time to render.

Figure 2.1: A Simple Ray Tracing Illustration[2-1]



Since a sphere can serve as its own bounding volume, initial experiments with the shading processor used spheres as test objects. For nonspherical objects, additional intersection processors must be specified whenever a ray does intersect the bounding sphere for that object. For polygonal surfaces the algorithm solves for the point of intersection of the ray and the plane of the polygon and then checks to see if the point is on the interior of the polygon. If the surface consists of bicubic patches, bounding spheres are generated for each patch. If the bounding sphere is pierced by the ray, then the patch is subdivided using a method described by Catmull and Clark [10], and bounding spheres are produced for each subpatch. The subdivision process is repeated until either no bounding spheres are intersected (i.e., the patch is not intersected by the ray) or the intersected bounding sphere is smaller than a predetermined minimum. This scheme was selected for simplicity rather than efficiency.

The visible surface algorithm also contains the mech-

The solution that Nvidia, the world's leading manufacturer/designer of computer graphic processing units (GPUs), devised to allow ray tracing to be done in real-time was RTX: Nvidia's ray tracing graphics cards[2-3]. The reason RTX GPUs could trace rays in real-time lies within the architecture of the graphics card itself. RTX cards have a reserved chipset on the graphics card that is dedicated to calculate light ray bounces upon objects in an environment.

The chipset itself utilizes SIMD Flynn taxonomy (Single Instruction [across] Multiple Data) to trace rays in real-time. In the common computer, CPUs utilize a taxonomy that applies a single instruction to a single piece of data. This taxonomy doesn't benefit the performance, but it instead provides *generalization* or *universalization* since applications themselves are general and need to be able to be ran on a wide range of systems (it has to be compatible with every user's computer!). However, for specific algorithms, SIMD may be a necessary approach in order to get the required performance. Basically, the rays' trajectories are calculated on dedicated hardware aside from the GPU's main CUDA cores (Nvidia's own special multicore architecture used for refreshing the image on your display) and the tracing algorithm, once performed on a vectored computational structure, has a derivative behavioral time order; that is to say that the computational time to trace the rays decreases significantly to a point at which we can trace them within the application without needing to keep rendering the scene. This is extremely useful for an application of which includes dynamic environments with moving light sources. One thought can be drawn from this solution: can we apply and dedicate other -rather intensive or performance-selfish- algorithms to other separate SIMD devices? These vectored architectures could be separate chipsets on a single card, or individual monolithic cards across different PCI slots on a computer motherboard working in parallel. As stated by Arthur Appel in 1969, ray tracing algorithms are very time consuming, taking 1000's[2-5] of times longer than a simple wire frame drawing, and in this duration of time, about *half of it*[2-5] is used to determine point-to-point correspondence (BVD[2-7]) of the projections onto the screen aside from *rasterization* (the primary technique used to draw a 3-dimensional object onto a 2-dimensional display to shade and wrap pixels before final redisplay).

Before real-time ray tracing was possible, Professor Jim Kajiya, one of the most influential

FIGURE 25.1

Synthetic cloud with Lattice Boltzmann lighting.

Table 25.3 Execution time for lighting the 256×128^2 cloud model.

Platform	Time (Seconds)
Intel i7 X980	1170
Intel i7 X980 with SSE	281
Intel i7 X980 with SSE and four-way threading	100
NVIDIA GTX 480 naive kernel	167
NVIDIA GTX 480 final kernel	6

Table 25.4 Execution time for lighting a 128^3 cloud model.

Platform	Time (Seconds)
Intel i7 X980	292
Intel i7 X980 with SSE	70
Intel i7 X980 with SSE and four-way threading	26
NVIDIA GTX 480 naive kernel	41
NVIDIA GTX 480 final kernel	2

Figure 2.2: The Illuminated Lattice-Boltzmann Cloud

computer graphics researcher of all time, stated "ray tracing is not slow - computers are." It is this “half-the-time” that is alleviated through the use of SIMD vector computing via the separate chipset on Nvidia’s RTX GPUs. This is supported by the *rendering equation*[2-7] illustrated by Kajiya:

$$I(x,x') = g(x,x')\left[\epsilon(x,x') + \int_S \rho(x,x',x'')I(x',x'')dx''\right]. \qquad (1)$$

Figure 2.3: The Rendering Equation

where:

$I(x,x')$	is the related to the intensity of light passing from point x' to point x
$g(x,x')$	is a "geometry" term
$\epsilon(x,x')$	is related to the intensity of emitted light from x' to x
$\rho(x,x'x'')$	is related to the intensity of light scattered from x'' to$x$ by a patch of surface at x'

This equation describes the light intensity (the shade *per se*) of a point x from an initial source x’ and a collection of light bounces from another source point(s) x”. The important part of this equation to notice, when it comes to the computer science aspect of it, is the integrated portion. Light intensity computation is originally linear in behavior (applying a constant effect to n pixels [rasterization]) but the integral in this equation describes the complete summation of infinitesimally-small (realistically is a fixed number of iterations/degrees of bouncing in computer science determined by the programmer’s discretion) light bounces from a source surface to the destination surface. This summation, when viewed in the light of real-time applications with large amounts of light sources and objects to bounce off of, can get very time consuming because of the rigorous use of the equation for each point in the 3D space in an environment. With Nvidia’s RTX GPUs, the CUDA cores compute the rasterization of 3D models and, in parallel, compute the final lighting of points in the environment from ray bounces. In other words, in terms of the equations, the iterative integral section of the rendering equation is satisfied by vector processing (SIMD architecture) while the rest of the scene is established through the GPU’s processor (also of SIMD classification). Currently, I believe we need to push the innovative energy behind the idea of “small supercomputers” -built for specific tasks, yet accessible by consumers, and supported by programmers. Ray tracing is just one of many intensive algorithms that would run in the background to achieve a quality application; clip checking (model “clipping”), cloth physics, liquid physics, multisampled anti aliasing, and others are functions that are intensive within a 3D real time application that could use further hardware performance. Outside of software functions, there are system functions that could benefit from this style of hardware- system functions for self-learning AI devices is a modern example of such a system. What about other intensive runtime components of a real-time application? Can we devote AI behavior to a single piece of hardware to give a smarter capacity to AI in games? Anti-aliasing and clipping to another? The one thing prior to real-time ray tracing that held it back was hardware support, so with other intensive computer algorithms, perhaps we should continue to broaden our possibilities to ensure a smooth experience in a new age of photorealistic applications. Or is this too early an innovation within the market?

References (See Annotations)

[2-1]: *Practical Ray Tracing in C: Buch* | Lindley, Craig A.
[2-2]: "Chapter 9: Ray Tracing." *New Advances in Computer Graphics* | Proceedings of Cg International '89
[2-3]: "Introducing the NVIDIA RTX Ray Tracing Platform." | NVIDIA Developer
[2-4]: "What's the Difference Between Ray Tracing, Rasterization?" | NVIDIA Blog
[2-5]: *Computer Architecture*: Chapter 4 | Elsevier Inc.
[2-6]: GPU Computing Gems: Emerald Edition | Hwu, Wen-mei.
[2-7]: *SIGGRAPH '86 Conference Proceedings* | Evans, David C.

Chapter 3
The Scientific Market

Pfeil, Alan R.

Introduction

Aside from sole demand or consumer enthusiasm, the capabilities and willingness of the software and hardware companies that would support and implement these growths are small in number; not necessarily that they aren't *capable* programmers, but that there isn't enough *experienced* vector programmers to help foster further innovation and complete the bigger picture. Once mentioned by Dr. Chi Yan Leung, a CSE computer architecture professor at UCM, that there isn't enough familiar vector programmers in the field able to program and create self-learning AI devices and other SIMD architectures. Funnily enough, this becomes evident when searching for "vector processor programming" online; even *Wikipedia*'s vector processing webpage is only a thousand words in length; there just aren't enough skilled programmers for these processors such that enough gets taught.

Assessment: Vector Programmers, GPU Performance, and Consumer Enthusiasm

Vector processor programming isn't a separate programming language, but is rather a separate technique for programmers to abide by. For modern SISD[2-5]/MIMD[2-5] (Single Instruction [on] Single Data & Multiple Instructions [on] Multiple Data) processors -like the ones you have in your own personal computer- programmers share the application load across the multiple cores and threads that your processor supports, and if you have a different amount of cores/threads, the program will dynamically adapt to your CPU's own pipeline (it's order of execution per se). However, with vector processors, each processor, core, and/or "thread" is different and behaves differently with varying run times for separate processes. This is a burden for the programmer, for they have to figure out a sufficient pipeline when compiling their code that can work efficiently on the vector processor they're working on. Levesque and Vose exemplify vector programming with Nvidia's GPUs[2-6]; applications could not always be vectorized and/or parallelized automatically by the compiler. The developers themselves have to do something for better performance in these cases and that comes in the form of CUDA core programming. Since supercomputers utilize vector processors, think of how it would be if a programmer would code for a supercomputer -built for specific tasks- in contrast to how they code for normal computers on a general application; the process is nuance, and the support is narrowly limited. Therefore, most people find little use in learning such a skill. However, with today's increasing popularity with intense big-data systems like self-learning AI, Google's big-data algorithms, and ray tracing (real-time or rendered), SIMD[2-5] devices are seeing more and more use hence the rigorous nature of tackling a lot of data with a lot of instructions. Take Alexa or Google Home for example, their algorithms try to learn about what you're telling it to do or what you're saying and they have to respond by the time you would expect an answer. In order to search the web and learn, these AIs need to explore a vast amount of web pages regarding your topic to supply an answer in real-time sufficiently and no current algorithm on a MIMD[2-5] processor could support such a quick and thorough response as such from one on a SIMD device; therefore like GPUs, small SIMD devices are included in these machines such that they can search through multiple pages at once rather than going through them sequentially.

In a discussion with Nvidia's CEO, Jen-Hsun Huang, Huang states that RTX's lack of success in the sales department was a result of them releasing high-end flagships first before other mid or entry-tier RTX platforms[3-2]. However, looking back towards previous flagship releases by Nvidia, their team released the GeForce GTX 1070 and 1080 on launch of the 1000 series cards. These two GPUs are classified as "high" and "enthusiast" tier by WCCFTech[3-2] and the common PC gamer. The launch price for these previous high-end cards were from $380-$500 USD, respectively. For the RTX platform, on launch, the RTX 2080 and 2080ti were being shipped out (September 20, 2018) for $800-$1200 USD. The RTX 2080 and 2080ti are classified as "enthusiast" and "ultra-enthusiast" tier, but this is only a "step" increase in launch tiers with a huge increase in price when contrasting the launch prices between the 900 series and the 1000 series: the GTX 970 was $330 USD on launch and the GTX 980 was $550 USD on launch. This displays a $50 increase between high-tier cards, with a $50 decrease with the enthusiast-tier cards. Now looking back at the RTX platform, a $420 increase is observed for the enthusiast-tier cards which is not proportional to the *base* performance gain on general real-time 3D applications. It is true that we could expect a greater cost for technology that is seperate from the previous generations (RTX vs. GTX flagships) hence real time ray tracing is such a rigorous feat. However, consumers wanted a proportional performance gain over real-time ray tracing capabilities ***because*** of the low amount of support for RTX's main feature/selling point. Before getting to RTX's real-time ray tracing support, it's important to illustrate the performance difference between the 1000 series and the 2000 series:

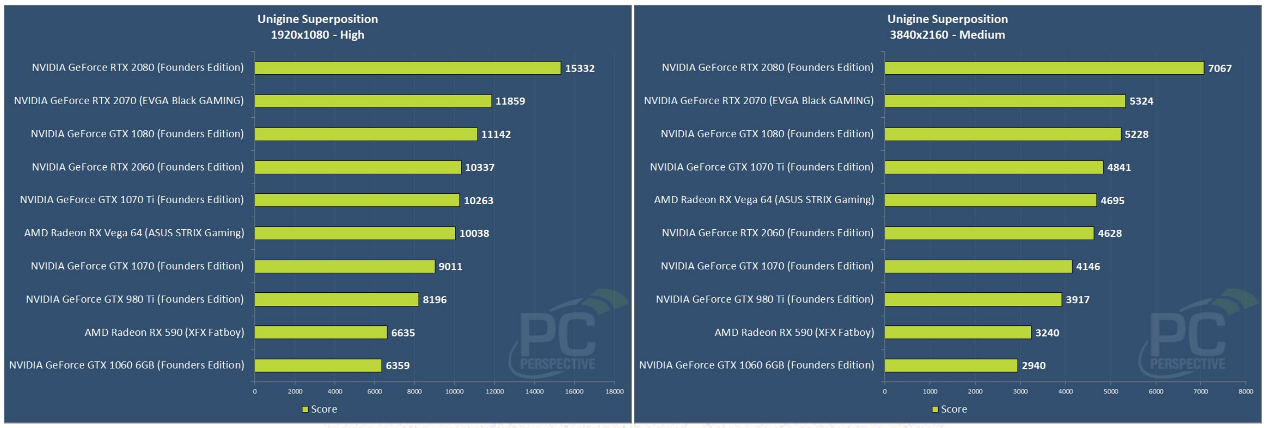
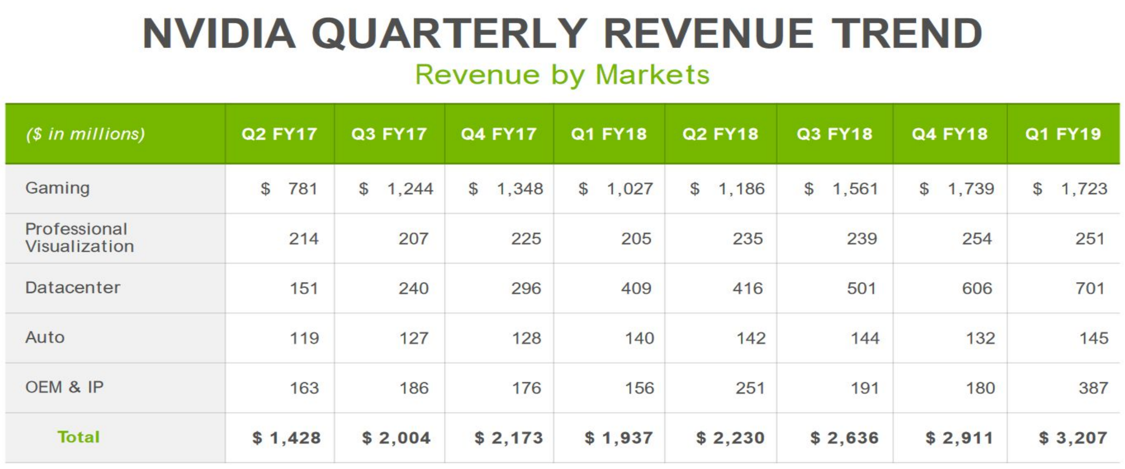


Figure 3.1: Unigine Benchmark Across Modern GPUs

[illegible]



NVIDIA QUARTERLY REVENUE TREND

Revenue by Markets

($ in millions)	Q2 FY17	Q3 FY17	Q4 FY17	Q1 FY18	Q2 FY18	Q3 FY18	Q4 FY18	Q1 FY19
Gaming	$ 781	$ 1,244	$ 1,348	$ 1,027	$ 1,186	$ 1,561	$ 1,739	$ 1,723
Professional Visualization	214	207	225	205	235	239	254	251
Datacenter	151	240	296	409	416	501	606	701
Auto	119	127	128	140	142	144	132	145
OEM & IP	163	186	176	156	251	191	180	387
Total	$ 1,428	$ 2,004	$ 2,173	$ 1,937	$ 2,230	$ 2,636	$ 2,911	$ 3,207

Within the graphics card benchmarking application Unigine, across full HD (1920x1080p resolution) and ultra HD (3840x2160p resolution) the RTX 2080 and the GTX 1080 have a considerable performance difference when testing the card's capabilities. However, with Nvidia's big market share being in the video game industry, modern graphically-intensive games may observe a less significant increase (if any) in performance (also due to game optimizations): *Figure 3.3: Battlefield V's Modern GPU Performance*

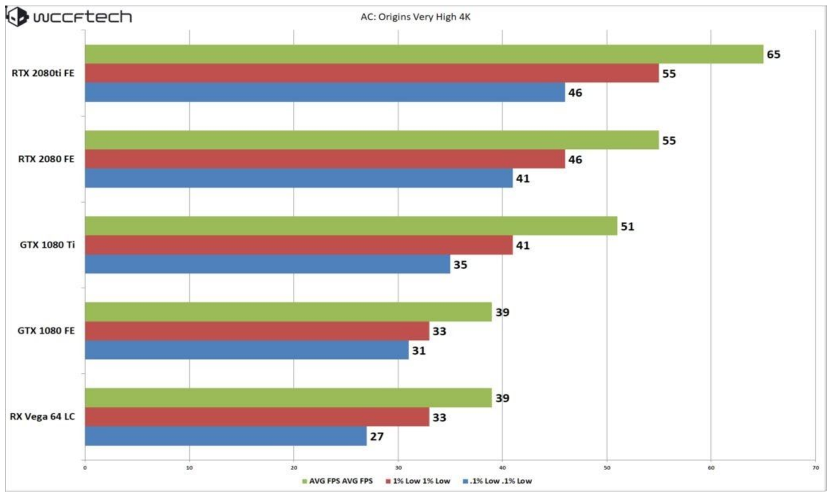
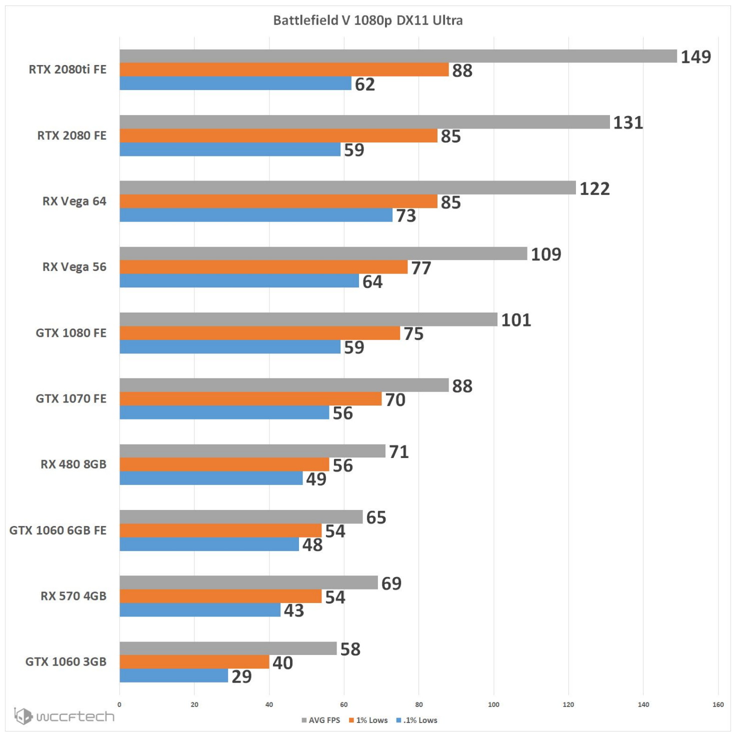
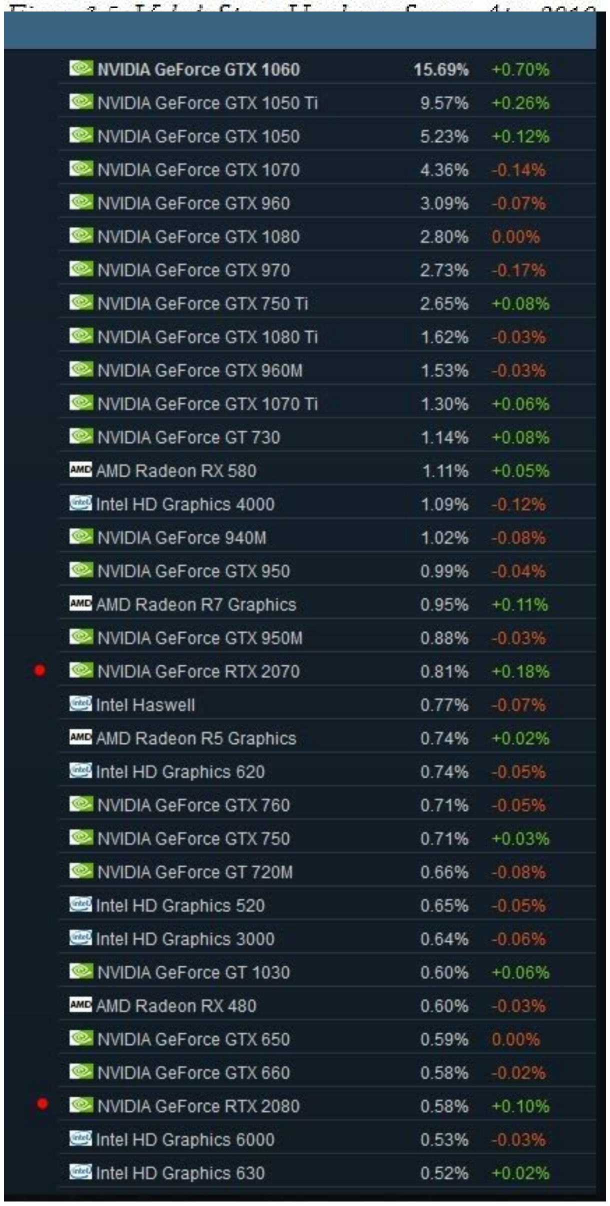


Figure 3.4: AC Origins Performance w/Modern GPUs (no ray tracing support/functionality)



Again, in spite of the performance gain -whether or not in games or on benchmarks- the performance increase is seen to be 'not worth' its difference in price regardless of the added on technology inside of it. This can be qualitatively observed by comparing Gamespot's hardware survey from February of 2018 when the site claimed that 14.2% of Steam users (a primary digital games store/platform) use Nvidia's GTX 1060[3-4] released back in 2016. If you were to visit the official Steam hardware survey from March of 2019, the same trend can be found: the GTX 1060 is still the most popular card used but even more so than before with a 15.69% (figure 8) ownership rate among Steam users[3-5]. Therefore,
the second proposed reason, brought forward by WCCFTech, seems to be a more relevant one: "the main feature or RTX itself. NVIDIA had promised many gaming titles to use their highly anticipated Ray Tracing and DLSS tech but currently, only three games exist that utilize their DLSS technology and two games, Battlefield V and Metro Exodus, which use Ray Tracing. Sure the Ray Tracing tech helps a lot in enhancing the visual fidelity but Ray Tracing also impacts performance. In each title tested, Ray Tracing means a major hit to FPS so users would have to trade in performance for quality and in a title such as BFV Multiplayer where FPS matters, Ray Tracing can take away the fun from the game unless you tone down the resolution to somewhat playable levels." In other words, it's not what users collectively *needed* nor *wanted* despite the future potential gained in achieving real-time photorealism; it was *too* big of a step for consumers to jump on board. Their gauge for purchasing an RTX card was founded upon the amount of entertainment versus cost. When a person wants to be entertained they can get a robust version of that same product and they would be fulfilled. Especially when the robust edition is a fraction of the price with almost the same power to entertain.

On a more recent note, the main reason why computer gaming enthusiasts aren't jumping onto the RTX platform has earned an interesting solution: Nvidia's ray tracing technology was recently released on a popular open-source gaming engine by the name of *Unreal Engine*. Considering the accessibility



	NVIDIA GeForce GTX 1060	15.69%	+0.70%
	NVIDIA GeForce GTX 1050 Ti	9.57%	+0.26%
	NVIDIA GeForce GTX 1050	5.23%	+0.12%
	NVIDIA GeForce GTX 1070	4.36%	-0.14%
	NVIDIA GeForce GTX 960	3.09%	-0.07%
	NVIDIA GeForce GTX 1080	2.80%	0.00%
	NVIDIA GeForce GTX 970	2.73%	-0.17%
	NVIDIA GeForce GTX 750 Ti	2.65%	+0.08%
	NVIDIA GeForce GTX 1080 Ti	1.62%	-0.03%
	NVIDIA GeForce GTX 960M	1.53%	-0.03%
	NVIDIA GeForce GTX 1070 Ti	1.30%	+0.06%
	NVIDIA GeForce GT 730	1.14%	+0.08%
	AMD Radeon RX 580	1.11%	+0.05%
	Intel HD Graphics 4000	1.09%	-0.12%
	NVIDIA GeForce 940M	1.02%	-0.08%
	NVIDIA GeForce GTX 950	0.99%	-0.04%
	AMD Radeon R7 Graphics	0.95%	+0.11%
	NVIDIA GeForce GTX 950M	0.88%	-0.03%
●	NVIDIA GeForce RTX 2070	0.81%	+0.18%
	Intel Haswell	0.77%	-0.07%
	AMD Radeon R5 Graphics	0.74%	+0.02%
	Intel HD Graphics 620	0.74%	-0.05%
	NVIDIA GeForce GTX 760	0.71%	-0.05%
	NVIDIA GeForce GTX 750	0.71%	+0.03%
	NVIDIA GeForce GT 720M	0.66%	-0.08%
	Intel HD Graphics 520	0.65%	-0.05%
	Intel HD Graphics 3000	0.64%	-0.06%
	NVIDIA GeForce GT 1030	0.60%	+0.06%
	AMD Radeon RX 480	0.60%	-0.03%
	NVIDIA GeForce GTX 650	0.59%	0.00%
	NVIDIA GeForce GTX 660	0.58%	-0.02%
●	NVIDIA GeForce RTX 2080	0.58%	+0.10%
	Intel HD Graphics 6000	0.53%	-0.03%
	Intel HD Graphics 630	0.52%	+0.02%

of the engine, and the sheer outreach to a wider range of programmers, this update should prove to be an interesting point of observation upon the implementation by current programmers whether or not they choose to use the technology- or feel inclined to use it. If they are so inclined, support for this alternative rendering method may be exponentiated to a point that RTX may be viable to the common gaming enthusiast. Straying away from Nvidia's primary market share, SIMD's residual implications in the future still wield some form of gravity that other hardware components have to offer. SIMD architecture isn't supposed to be -solely- a faster iteration of computer hardware, it's supposed to be a game changer that unlocks more potential for other algorithms and software to be efficiently (or even practically) used. Thereby, in an industry that's only growing bigger and bigger in the future, we will need to implement more 'specialized' machines that will be able to do specific tasks efficiently, and this can -currently- only be achieved via SIMD architecture and vector programming.

Overall it is important to appreciate and understand the complexities of simple scientific achievement; innovation is left to students and teams. Larger developers commonly pass the writ of responsibility to smaller companies until that innovation is more "fleshed-out." With ray tracing, we can observe a stepping stone towards achieving real-time photorealism; something that may be implemented as a common (primary or alternative) method of rendering the next frame with visual conservation. It is an ultimate goal of ray tracing to be adopted among programmers outside of the small R&D "breakthrough" teams that are early adopters of the technology, trying to prove its capabilities. In light of this, Unreal's published real-time photo-realistic application video[3-1] serves as an example of just how far we've come without the need for real-time ray tracing. With this example of implementing an engine's core features (ray tracing being one of many) we can concur that the absence of such graphics could be a result of the aforementioned complexities: the aspects of art (could include photo-realistic features or not), the careful triple-A developers, the reluctant/uninformed programmer, and the elusive consumer demand for photo-realism. There are many reasons for such a slow adaptation, and they're all valid delays. The addition of innovations are slow as a matter of human nature as much as they are scientific.

References (See Annotations)

[3-1]: *UNREAL ENGINE 4 - Photorealistic Graphic (2019)* | Garth, Joe
[3-2]: "NVIDIA GeForce RTX Sales Lower Than Expected as Revenue Falls 45%." | Wccftech
[3-3]: "Discrete Desktop GPU Market Trends Q3 2016: GPU Shipments Hit Two-Year High." | AnandTech
[3-4]: "Here's The Most Popular PC Hardware Among Steam Users (February 2018 Update)." | GameSpot

PAGE BREAK INTENTIONAL

Summary Assessment (Team Conclusion)

This conclusion is a team piece. Everyone adds chunks to put it all together.

Strong parallels between our sections:

- traditional computing methods are too slow and can be improved
- RTX (ray tracing) significance versus sales failure
- Real-time photorealism; There are implications to this
- RTX's success shows proof of concept for SIMT computing; has potential

● Opening paragraph

Issues/aspect (2-8 paragraphs, depending on team size)

1. Examine issues and aspects or considerations

Close:

Within the scope of this document I hope to raise a cause and effect relation between achieving scientific feats such as ray tracing and real-time photorealism and the scientific market. The market is its own entity that is arguably more/less complex than the scientific world and its strives. Simply put, the dissonance between chapter 1's assessment (that ray tracing isn't necessarily demanded by the common user [whether they'll want it later or not]) mixed with computer science's cutting edge solutions in chapter 2 (real-time ray tracing using SIMT), creates cacophony in what I like to call the *scientific market* as a whole. Stunning innovation and technology doesn't inherently imply a market success. True, somewhere along in the future if virtual reality starts becoming close to simulating real life, we'll need real-time ray tracing technologies much more than now. If this future carries out, ray tracing will get its "big break" if this is where consumer enthusiasm is placed. However, given today's current standings in the computer science and video gaming world, the average consumer is not interested in real-time ray tracing since current establishments can already simulate these effects to a fair degree with good performance. Thus, we need to think outward when considering other uses for SIMD (or SIMD-like) architecture. For industrial architecture, animation rendering times, and big data algorithms, Nvidia's real-time ray tracing solution could translate to a degree in solving the prior's problem(s). As a computer scientist and engineer, we are taught to implement things as efficiently as possible, exploiting both software (algorithm run times, optimization) and hardware (pipeline/parallel scheduling, cache preempting, loop unrolling) qualities of computer programs. In order to support and further this rule, we ought to look at what else we can vectorize when it comes to other forms of computer science problems and see the potential they might unlock in securing an innovative future.

.

Acknowledgements

We would like to personally thank Professor Nuno Sena for support and creation of this document including oversight and formation.

Appendix

Presentation

https://docs.google.com/presentation/d/1jesf1HG7UPaK8fysU3ceZPBOTrKhcJ6XlzHkm5ddPyk/edit?usp=sharing

SIMD Mode Graphic:

Single Instruction on Multiple Data shown on an array (A0-A7). An operation is performed on all A0-A7 at one by hardware architecture rather that iterating from 0 to ‘n - 1’, where ‘n’ is the length of the array. The left example (SIMD) is 1 constant operation while the right example (SISD) is 8 constant operations.

The Rendering Equation:

$$I(x,x') = g(x,x')\left[\varepsilon(x,x') + \int_S \rho(x,x',x'')I(x,x'')dx''\right]$$

I(x, x’) is the related to the intensity of light passing from point z' to point x.
g(x, x’) is a geometrical term.
ε(x, x’) is related to the intensity of emitted light from x' to x.
ϱ(x, x’, x”) is related to the intensity of light scattered from x" fox by a patch of surface at z'. The integral is taken over the union of all points x, x’, and x” ($S = \cup S_i$), or in terms of ray tracing, the bounds are determined by the number of rays of which we bounce across the points described.

Properties of Light Graphic:

The properties of light actuate for every ray being traced. Properties of light considered for traced rays in computer graphics are: refraction, reflection, scattering (local/impact illumination), and shadowing/shading.

Rendering Graphic:

This simple graphic was used to give a sense to the reader of the time, and busy-work, it usually takes to render and rasterize simple videos into a 2D wrapper.

Annotations

Chapter 1:

Chapter 2:

(2-1) Lindley, Craig A. *Practical Ray Tracing in C: Buch*. New York: Wiley, 1992. Print.

Provides the actual implemented code for ray tracing in a widely-used programming language which compares to the *rendering equation* by Kajiya[2-7]. With the code, Craig adds annotations and explanations for each section of the code to help walk through parts of the function. In parallel to the code, Craig includes an illustration of the code (p.1) to represent how the code tackles the 3-dimensional issue (does not fully cover rasterization techniques, but includes what needs to be done before/as rasterization takes place).

(2-2) Earnshaw, Rae A, and Brian Wyvill. "Chapter 9: Ray Tracing." *New Advances in Computer Graphics: Proceedings of Cg International '89*. Tokyo: Springer Japan, 1989. Internet resource.

Ray tracing within the 3D computer world *without* the effects of real time dependency highlight the taxing nature of the algorithm, but more importantly, it provides an idea towards the other overlapping fields ray tracing reaches- animation for example. In the animation industry, smaller animation studios/single animators may want to create animations using ray tracing, but even for a still frame, renderings of scenes (a *single frame*) with ray tracing effects can take up to hours to finalize at a time. This establishment emphasizes the impact on applications that are not real-time yet benefit from having ray tracing. Important to note this document was created in 1989 and current computers could render these frames in less time, however from the data in resource [2-6], graphics cards from 2010 still took a considerable amount of time to render a single cloud in a frame with ray tracing enabled. In light of this, it helps to show that the algorithm itself hasn't changed overtime, and even though our hardware has, the sheer performance of the hardware still isn't satisfactory in developing rendered frames within a *convenient* amount of time compared to those rendering times demanded by real-time applications. In order to have multiple Lattice-Boltzmann-like objects in a scene to be updated in real time, it is up to the hardware divisions of the field to come up with a *solution*[2-3] to get scenes drawn faster whether it includes creating a whole new type of graphics card or SIMD device so that the proper operations can be done efficiently; as described by Kajiya[2-7]: "algorithms aren't slow. Computer are."

(2-3) "Introducing the NVIDIA RTX Ray Tracing Platform." *NVIDIA Developer*, 10 Oct. 2018, developer.nvidia.com/rtx/raytracing. (https://developer.nvidia.com/discover/ray-tracing)

A formal introduction to RTX by Nvidia. Taking this introduction with a grain of salt may be beneficial since the source has implicit bias towards the success and significance of RTX technology: this is an official Nvidia blog "statement" of sorts to the public to advertise and explain how ray tracing works as its implementations upon field of computer graphics. Specifically, this source was used to provide details and qualities of ray tracing, the "locomotion" of the ray tracing algorithm (using SIMT style programming/hardware within the GPU), and a visual aid for the process of tracing rays. The page also describes the rasterization process, which is vital to understanding how graphics are processed for final "shipment" when the screen is updated or "drawn" after a specified "wrapper" has been made. A wrapper is a term mostly used in image processing in computer science where a scene (usually 2D) has all of its pixel gradients or color code "saved". By this, I mean that instead of having a bunch of data being in/outputted through the screen all you have is superficial data being in/outputted. When you think of a wrapper think of a still image that has no more value than you see; it is a combination of pixels with each their own color code which results in an image on the screen. Nvidia puts it as such: "is a technique used to display three-dimensional objects on a two-dimensional screen." Now, rasterization is the process which you take a 3-dimensional environment, and you take a snapshot of where the environment's *camera* is at. During rasterization you make sure all shading, graphics, and after effects, are applied at the current time and you then neglect those effect's data when you save the scene's shading; the pixel only needs to know it's color, it does not need to know *what* it's coloring-in nor what shape it's making, etc. Once the scene is rasterized into a resulting image it can then be manipulated as such or inputted into the GPU's CUDA cores where it will then be eventually displayed on your monitor. This process as a whole should help you understand the implied difficulty associated with making ray tracing achievable in real-time because these "rasterized wrappers" are made 30-60+ times a second when updating your monitor (fps [frames-per-second] depends on monitor's refresh rate in Hz [commonly 24.6Hz, 30Hz, 60Hz, 72Hz, 120Hz, 144Hz, etc.] but that is irrelevant for the time being). So in order to satisfy your monitor's refresh rate and achieve a smooth visual stream you have to make sure all algorithms, effects processes, and all necessary general program calculations are completed and ready to "ship" as a wrapper in order for it to update the screen promptly. Therefore, the algorithm must have a run time that will satisfy this demand. If it does not satisfy the demand then you'll have framerate issues for you application. Depending on how delayed your algorithms is completed will determine just how 'slow' your computer will update the image. In the case of ray tracing, since the algorithm would take minutes to days to make a wrapper of a scene you would expect an fps between 0.016 fps (1 frame every minute) to 0.000012 fps (one frame every day). Nvidia remarks "Rasterization is used in real-time computer graphics and while still computationally intensive, it is less so compared to Ray Tracing." and thus, in the realm of real-time applications (where you explicitly want to achieve *at least* 24.6 fps), it is observable that implementing ray tracing in real time isn't a possibility given current hardware solutions. Nvidia realized this and thus created a hybrid of rasterization and ray tracing. Nvidia describes their alternative approach in rendering as such: "...a technique that uses rasterization and ray tracing concurrently to render scenes in games or other applications. Rasterization can determine visible objects and render many areas of a scene well and with high performance. Ray tracing is best utilized for rendering physically accurate reflections, refractions, and shadows. Used together, they are very effective at attaining high quality with good frame rates." This description distinguishes the fact that rasterization and ray tracing aren't necessarily separate entities in Nvidia's RTX technology, but instead, RTX uses a hybrid of rasterization with ray tracing within an SIMT/SIMD architecture in order to get ray tracing achievable in real-time.

(2-4) "What's the Difference Between Ray Tracing, Rasterization? | NVIDIA Blog." *The Official NVIDIA Blog*, 1 Aug. 2018, blogs.nvidia.com/blog/2018/03/19/whats-difference-between-ray-tracing-rasterization/.

An important read to establish the difference between ray tracing in real-time and rasterization; why it's important to trace rays before/alongside rasterization to create a wrapper product (a processed frame in this case). Some of the images included in this article depict scenes that are rendered with ray tracing technology which aid in visibly seeing the potential in graphical fidelity that ray tracing can achieve (among many other graphics concepts and effects). In addition, this document also sheds light upon the working nature of Nvidia's RTX hardware. Nvidia explains and references ray tracing in the late 70's and mid 80's, including the use of Jim Kajiya's rendering equation$_{[2\text{-}7]}$ from 1969. Nvidia explains their hardware (RTX) is "a result of decades of work in computer graphics algorithms and GPU architectures." which is made apparent from the publishing dates of references [2-1], [2-2], and [2-7]. Nvidia tries to explain where they believe ray tracing is going with video gaming being the frontier of its implementation. In a general industry though, Nvidia believes ray tracing will be used by architects and designers in map making and model creation since their ray tracing technology takes care of carefully-tweaked realistic effects and makes realistic scenes look real in a simple manner. In other words, instead of needing to create multiple realistic effects for a scene to look real, you simply let ray tracing do its job in creating those effects for you *automatically*. After all, the main basis for visual effects starts with the manipulation of light in the first place. Nvidia eventually hopes in seeing their technology being used in video games with enough craft to achieve "movie-quality rendering" in real-time so that playing a game would perhaps be akin to playing an interactable movie. These wishes Nvidia had planned is a roadmap that takes time to "sit back" and see how it's implemented/used. You cannot expect major triple-A developers to switch their rendering methods (the only one, really) into a whole new one; it's seen as too risky for some big developers, so most likely we won't see huge enough implementations until independent teams can prove the safety and usefulness of ray tracing. Perhaps Nvidia already knows this and doesn't expect huge developers to go out of their way. Video game creation takes years of effort by huge teams just like movies do, so Nvidia wouldn't expect "big-time" upgrades in fidelity until at least 2 years after release (aside from contractually obligated studios that have worked alongside Nvidia to produce games with ray tracing technology). As of present, we're still waiting for more teams to use Nvidia's rendering methods to produce photorealistic games. However, referring to annotation [3-1], Unreal Engine has released update 4.22 which adds ray tracing hybrid rasterization techniques to the engine. Hence this engine is widely used by independent and triple-A studios alike, this release may be the next stepping stone in Nvidia's future predictions for ray tracing depending on if their technology will be used or not by developers on the Unreal Engine.

(2-5) *Computer Architecture*: Chapter 4: Data-Level Parallelism in Vector, SIMD, and GPU Architectures Elsevier Inc., 12 Nov. 2013, www.cse.msu.edu/~cse820/lectures/CAQA5e_ch4.pdf.

Flynn taxonomy and classifications of computer hardware: definitions and clarifications: SISD: Single instruction stream, single data stream; classic example is a MIPS processing unit (CPU); this type of computer architecture classification is very important to support dynamic/widely-varying programs. It is indeed the main vehicle behind processing most of all desktop applications hence SISD devices can 'modulize' software calculations. •SIMD: Single instruction stream, multiple data streams; vector architectures; this is the main classification of computer architecture that I cover; the most common use of SIMD architecture is found in a computer graphics cars to update pixels on a screen through rasterization techniques for real time 3D applications (running on software engines). Other examples include supercomputer central processing units (vector processors). •MIMD: Multiple instruction streams, multiple data streams; important, but limited use in the computer industry; not relevant to this paper. •MISD: Multiple instruction streams, single data stream; no significant commercial implementation as of present. These classifications define the hardware architecture in my research; for clarification and communicative purposes. It is also further described by Arthur that the amount of computing 'power' (running time) is much greater (1000's of times longer) in ray tracing than tracing/drawing a "wire frame" (a 3D model [data structure] consisting of vertices and vectors to make up the shape). These wire frames are 'higher' order pieces of data that help establish the final wrapper during rasterization$_{[2-3][2-4]}$. These wire frames are lost after the wrapper is made (not lost within the program, because obviously shapes and figures will most likely need to be redrawn the next scene depending on where the camera moves within the 3D environment) but the wire frames' data is not present in the resulting image; we don't care where the wire frame's vertices are, we only care if it's *there or not* in the resulting image on our monitor. Considering this distinction between how long it takes to draw a wire frame, it can be grasped the idea -the runtime- of the ray tracing function(s) versus other drawing functions in real-time applications.

(2-6) Hwu, Wen-mei. GPU Computing Gems: Emerald Edition. Elsevier, 2011

Lighting a cloud with the Lattice Boltzmann ray tracing method (384). The graphics card used in the lighting of the cloud is an Nvidia GTX 480. The GTX 480 is a dated graphics card (new and relevant at time of publication) without real time ray tracing capabilities. Comparing the computation time for lighting the cloud, 6-167 seconds, and current-day ray tracing technology found within an RTX 2080, full runtime applications like Metro Exodus running at least 30 to 60 frames a second (at least 0.0333 - 0.0167 drawing time) with numerous objects/light sources, according to my calculations, at least a 180x-10,000x performance increase), it can be evaluated that dedicated vector hardware can be effective in producing strenuous algorithm results "quickly." The pseudocode for the Lattice Boltzmann lighting model applies n directions (degrees) of bouncing to an object with density i, j, k. Within a real-time application where there are multiple objects, m, and potentially multiple light bounce iterations, it can be observed that the final kernel time is put under a lot of workload such that a polynomial algorithm, like the Lattice Boltzmann method, *appears* to have a time order of a higher degree. It is this lighting that would be used by Nvidia's ray tracing hybridization of rasterization[2-3][2-4] in order to create scenes with multiple Lattice Boltzmann clouds and carefully-crafted 3D environments, so the drawing times for the single cloud using a GTX 480 seems worrying without proper dedicated hardware to support the algorithm's frequented use. This trouble was Nvidia's whole idea behind creating RTX; a GPU that could utilize the ray tracing algorithm/rendering equation in real time. It is once, and only once, this form of lighting can be achieved in real time can we start to implement photorealism in 3D applications like video games, animation, concept modelling, architecture, movies and movie editors, etc. with more simplicity, less intention, and better accuracy.

(2-7) Evans, David C. *SIGGRAPH '86 Conference Proceedings, August 18-22, 1986, Dallas, Texas*. Association for Computing Machinery, 1985.

David Evans highlights Jim Kajiya's rendering equation (1) in his publishing at the SIGGRAPH 1986 computer graphics conference, displaying the explicit nature of the algorithm's run time on common hardware; the common consumer-grade computer (running SISD-classified processors and simple SIMD graphics cards). In comparison with big animation companies that own their own supercomputers to render frames, when ray tracing is executed on a consumer computer, on the CPU or dedicated graphics hardware, the runtime is considerably longer- regardless of the computing power of the consumer's hardware. This is because SISD machines operate by one calculation at a time to one piece of data[2-5] and within older GPUs, ray tracing algorithms simply take too long to draw the next scene in real-time; it must wait until rays are calculated and traced to draw the next frame of the scene (see [2-4]). The problem is that we want to be able to trace the rays' movement using SIMD architecture (since the rendering equation is repetitive with similar operations) so that the next wrapper[2-3] can be queued up for final display on the monitor. Kajiya coins that "ray tracing is not slow… computers are." which is a conclusion we would like to draw with the given boundaries we're pushing in the computer field; hardware is going to have to keep up with the demand of the software and the people.

***Chapter 3*:**

(3-1) Garth*, Joe, director. *UNREAL ENGINE 4 - Photorealistic Graphic (2019). YouTube*, Quixel, 29 Mar. 2019, youtu.be/zKu1Y-LlfNQ.

**Last name spelled as phonetically interpreted*

This source produces and serves as a visual aid in understanding the current-day state of real-time photorealism in 3D applications and the already-existing potential within present gaming engines. The video shows an example of an application built with the Unreal Engine (4). Many modern games today use this engine because of its opened source and accessibility. It is crucial to note that the demo was ran on a single Nvidia GTX 1080 Ti which is a mainstream (although high-end) graphics card of early 2017. This means the common application user has access to real-time photorealism without needing alternate hardware. This also means that the demo ***did not*** include ray tracing in its scene although now possible with Unreal Engine's 4.22 version release (this does not, however, disclude possible needs for alternate hardware for other atomic devices [such as self-learning AI machines/devices]). This highlights the aspect that real-time photorealism can be achieved without ray tracing due to the nature of crossing the uncanny valley; you wouldn't be able to distinguish a scene as real or 'unreal' unless told specifically what was missing and showed a comparison. Even then, the human mind will see both results as "real" even given the explicit difference. Simply put: ray tracing is a means of adding another stepping stone in the realm of real-time photorealism while -at the same time- offering a method of dealing with performance-taxing (iterative) algorithms. Additionally, according to Garth*, summer tutorials for the Unreal Engine are being set up, which implies the fact that Unreal doesn't force these engine functions upon the user; the user doesn't necessarily have to be aware of the functions either. This supports the connection(s) made with last two paragraphs of this annotated research: that it is by the user's discretion whether to use these features or not. For artistic purposes this exists. For other users, whom are going for a realistic route, these functions may prove to be efficient in achieving what the user has in mind if they are only aware and willing.

(3-2) Mujtaba, Hassan. "NVIDIA GeForce RTX Sales Lower Than Expected as Revenue Falls 45%." *Wccftech*, Wccftech, 15 Feb. 2019, wccftech.com/nvidia-geforce-rtx-20-sales-lower-than-expected-45-percent-revenue-decline/.

(Figures 5 & 6)

The first reason: "NVIDIA stated that the reason for poor sales was due to them introducing high-end cards first. But I should remind you that with Pascal and Maxwell, NVIDIA also launched their high-end cards first. The GeForce GTX 1080 and GeForce GTX 1070 were introduced around the same time while the GeForce GTX 980 and GeForce GTX 970 were also announced and launched close to one another. The GeForce GTX 960 and GTX 1060 always launched later than the higher end cards but this time, NVIDIA had the edge by offering RTX 2080 Ti from the start which is a crucial flagship product in the Turing lineup. 'When we launched the 2070 and 2080, it was the first time we've ever launched a new generation where the only available SKUs were very high end. And in addition to that, the early boards that came out into the marketplace were the special edition and the overclocked versions. And the MSRP versions didn't show up for some short time after, couple of months after. And so the conditions weren't ideal, if you will. We weren't able to launch into the mainstream segment with 2060 for all the reasons that I think everybody understands now. And so I think that the situation wasn't ideal. When you take a look at our situation now, every single graphics card had the best performance at its price point and it remains so today.' *The second reason:* NVIDIA had promised many gaming titles to use their highly anticipated Ray Tracing and DLSS tech, but currently, only three games exist that utilize their DLSS technology and two games (Battlefield V and Metro Exodus) which use ray tracing. Sure the ray tracing tech helps a lot in enhancing the visual fidelity but ray tracing also impacts performance. In each title tested, ray tracing means a major hit to FPS (Frames Per Second) so users would have to trade in performance for quality and in a title such as BFV Multiplayer where FPS matters, ray tracing can really take away the fun from the game unless you tone down the resolution to somewhat playable levels. "And I think that right out of the box, it delivered excellent performance. It is true that everybody was hoping to see more games with RTX on day one. But it's such a new technology with ray-tracing and AI for image processing that it's only really possible to make available with new games, which is tied to the schedules of new games. And now they're starting to come out. Battlefield 5, Metro Exodus, I think that the reviews from this week are just spectacular." With DLSS and Ray Tracing combined, users can indeed get higher playable framerates but DLSS has shown to cause image artifacts and in some cases, blurry image quality than standard TAA which may lead to poor game immersion. Plus, we still haven't seen Shadow of The Tomb Raider, and many other titles which were promised by NVIDIA to implement ray tracing and DLSS tech. Given the high bar set by Maxwell and Pascal before it, Turing just doesn't seem like much of an update in terms of pure raw power (minus DLSS and RTX features)."

(3-3) Shilov, Anton. "Discrete Desktop GPU Market Trends Q3 2016: GPU Shipments Hit Two-Year High." *RSS*, AnandTech, 28 Nov. 2016, www.anandtech.com/show/10864/discrete-desktop-gpu-market-trends-q3-2016/4.

This source represents the market share of Nvidia from 4 pools of discrete GPU use. Professional (software development, architecture rendering, animation, etc.) and datacenter (big data crunching, database solutions, etc.) take *about* 15% each of Nvidia's market, which is dwarfed by the ~73.5% market share that computer gaming owns (Q3 2017). This is a crucial distinction to make when regarding the success -or lack thereof- the RTX platform; hence gaming shares such a large percentage of Nvidia's market, it becomes important to see the subtle reasons why it's failing in that department rather than why it's failing overall; it's less a scientific reason and more of a support/entertainment reason, which is a significant distinction to make for long-term achievement. It is also important to note the overall growth of discrete desktop (and mobile) GPUs in both production and units sold. In 2016, Nvidia launched a new GPU architecture patent named "Pascal" which used 14nm and 16nm transistors versus Maxwell's 28nm transistors (predecessor). Pascal is what you would call a "sandbagged" version of Maxwell, which means that Pascal holds Moore's law to its definition: that doubling transistor count (or cutting its occupational space in half) happens every year (or 2 years, recently). This will deem 1:1 in its performance boost to the hardware. Considering this, it's pretty rational from a consumer standpoint to assume a similar boost in performance should be seen in early 2018 with RTX's launch, but RTX wasn't the standard flagship boost that consumers were accustomed to; it was a branch off from Nvidia's hardware innovation where they would test something new to the market rather than adding more *horsepower* than "bang-for-your-buck." So it's also reasonable and understandable that RTX wasn't the *big* launch that a consumer would see it to be since the largest share of Nvidia's GPU sales didn't understand its place among other discrete cards.

(3-4) Thang, Jimmy. "Here's The Most Popular PC Hardware Among Steam Users (February 2018 Update)." *GameSpot*, Gamespot, 3 Mar. 2018, www.gamespot.com/gallery/heres-the-most-popular-pc-hardware-among-steam-use/2900-1798/3/.

Thang explores Valve's Steam hardware survey (the leading 3rd party computer gaming launcher) upon which users submit their PC specifications to the survey by discretion. Hence Nvidia's largest market share is amongst computer gamers, it makes logical sense to use Steam's "over-compassing" position as a lead media launcher platform to evaluate what most gamers own. Gamespot points out in the survey that Nvidia GPUs take up roughly 85.3% of users on Steam. It is crucial to note that 90.8% of the users use a DX12 (DirectX-12) viable graphics card *which means* that over 90% of the users on Steam are using modern GPUs, and about 85.3% of these modern GPU users own a modern Nvidia card that supports DX12 (the first line-up of DX12 cards from Nvidia is any card of the 900 series and up [neglecting a few other specific cards]). This means that, by my calculations, about 77.5% of users on Steam own a 900 series card or greater. In another form, 77.5% of users recorded having a GTX 960-GTX 1080ti or an RTX 2060-RTX 2080ti. However, it is noticed that the most popular GPU is Nvidia's GTX 1060 coming in with 14.2% user usage followed by the GTX 960 and the GTX 750ti with 12% and 13.2% usage, respectively. This means that the dominant GPUs seen in the PC gaming scene doesn't include a single RTX 2000 series card in the top 3 lineup even given the fact that Nvidia has released their RTX 2060 budget card by this point in time (since the X60 model is usually the most popular card in each series) and Nvidia's release of the RTX 2070 and 2080 back in Q3 of 2018.

Therefore, from comparing the numbers that Gamespot, and Valve's Steam hardware survey of February 2019, portray, it can be drawn that most PC gamers aren't switching over to the newest lineup of Nvidia GPUs.

(3-5) "Steam Hardware & Software Survey*" Valve*, Apr. 2019, store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam

Now it gets interesting when you compare Steam's February of 2019 hardware survey with today's current hardware survey in April of 2019. Even with the passage of time since Nvidia's RTX launch, including their budget RTX 2060 launch, the dominant GPU seen by a Steam user is the GTX 1060 again. But most importantly, this time, we see a growth with the number of owned 1060's. From February's 14.2% ownership, April sees roughly a 1.5% (1.49%) increase with a GTX 1060 ownership share of ~15.7% (15.69% to be more precise). Not only does this show that older card usage is growing within RTX's availability, but when you actually look at the share of GPU devices among users you see that the RTX 2070 is in 19th place with 0.81% usage and a growth of +0.18%, which is much lower than GTX 1060's growth between the last month. The 2080 is next in 32nd place with 0.58% usage and 0.10% growth from last month. So not only is the share of GPU ownership leaning towards last-generation cards, but the growth rate of users hopping on board to last-generation cards is significantly outnumbering the amount of being hopping on-board with RTX cards. You can even see a GTX 660 beating the RTX 2080 in ownership and the GT 730 beating the RTX 2070 when those two cards were released in Q3 2012 and Q2 2014 respectively. Although RTX growth rates beat those of the much older cards, they're still dwarfed by the 1000 series' growth. Therefore, from a standpoint of a computer scientist, it may be confusing as to why the RTX hasn't been vastly popular with high consumer enthusiasm considering the scientific feat. However, from a consumer standpoint, akin to [3-3], there wasn't a tangible enough reason to justify the RTX's launch price when it came to raw game performance boost.