REFINITIV STREETEVENTS

EDITED TRANSCRIPT

NVDA.OQ - NVIDIA Corporation - Special Call

EVENT DATE/TIME: APRIL 18, 2023 / 5:00PM GMT

REFINITIV

CORPORATE PARTICIPANTS

Aurelio Reis

Jeffrey D. Fisher *NVIDIA Corporation - EVP*

Robbie Jensen

Zach Lo

PRESENTATION

Zach Lo

The episode of the level up within NVIDIA this month, we're talking about Nsight Graphics, how to opt -- we've got Aurelio Reis, who is a Software Engineering Director of Graphics and Developer Tools. And to the end and guarantee will be good.

Aurelio Reis

Actually, a bunch of different tools and a specific purpose for game developers. And so what we urge developers to do is actually start with some something called Nsight Systems, which is more -- and so you can imagine you're submitting work to the GP but are you submitting it the most? These are algorithms tuned in a way, the GPU fed, so that is actually able to process most to find that out. And it will actually show you additional things like transfer information across the PCI by you. And if you need to back for debugging where you want to make sure that the scene is correct. So directness analysis and debugging from that perspective and then you can debug your performance, which makes sure that you're getting the performance that you want and understanding the reasons for why you might not be getting that performance.

Filers, there's GP Trace. Share your climate heat map. A bunch of these features exist within but they're all accessible directly from -- is used to identify crashes that might occur within your application, specifically related to -- I think a lot of people are familiar with many divide by 0 or -- and you'll get a crash and breakpoint where you can go and examine as well. GPU has become much more complex. And so being able to identify where that crash might occur is really important. I can show you a little bit about that in a bit. But one of the really important bits about that is that you can actually integrate the SDK within your application and then you can supply these markers so that you can identify where the crash might have occurred.

The Nsight Perf SDK is another Nsight product that allows you to, in real time, analyze performance from your application. And so it's just an SDK that you can integrate and then you can either dump the performance data for any given frame or there's even a real-time analysis mode that you can integrate. So you can see this just in real time without having to launch another tool like Nsight Systems or Nsight Graphics.

What's really cool about this is you can integrate that into a continuous iteration solution or a nightly testing solution. And so you can imagine the situation where instead of before, you might have someone check in a new level or a new model and suddenly the performance tanks. And you're not exactly sure why that might be. This -- you can actually use the Nsight Perf SDK to generate reports in HTML format, but you can just e-mail to the team on a nightly basis or a daily basis and be able to see exactly what the reasons might have been for that regression in performance. And so, oh, it's actually that model. And the reason why it was that model was because the triangle count was incredibly high or there's a new shade or material that was applied to it that wasn't optimized yet. So the performance needs to be improved for it.

Incredibly powerful tools, and they're all available to you. You can actually go download them from the nvidia.zone website right now for free. You just need to create an account. There are additional versions of some of these tools as well. The Nsight Graphics actually has a version called Nsight Graphics Pro, which provides additional low-level information that is more useful for professional developers that need access to that information. And if you do need access to that you should have a DevTech or a DevRel that you are in communication with. You can just reach out to them, and they can help you out or you can e-mail us at devtool@nvidia.com, and we can try to help you as well.

Robbie Jensen

I'll say on these 4 tools is that they really work together very harmoniously, I think that's how they were built to complement each other and feed into each other. An example that I always love to give is how NSIGHT Systems just gives you that one-shot look at like the entire systems performance and then something that you spot that like maybe your GPU is just spending a ton of time on this one frame. You can go into Nsight Graphics and examine why. So the advice that we usually give is to use all of them, together they are holistic development platform.

I think for more experienced developers who are already familiar with the tools, maybe they could be just developing their game and intuitively know that they have some sort of inefficiency that Nsight Graphics will just be able to snipe and alleviate like at a snap of your fingers. And that's great. But if you're a beginner developer and looking at all these tools, you could be thinking like where do I start? How do they plug into each other. What's the first thing you look at?

Honestly, of course, every situation is going to be different. But a lot of the times, the tools will tell you where to start. Like if you -- the way that like systems and graphics work is that they attach on to your game like an executable and they'll capture like a minute of your gameplay or something. And when you load up a timeline of performance in hardware throughput, there will be a flag on something that clearly is interfering with your performance. And from there, you can follow like a cookie trail to debug and optimize it. So really, together, it's like a -- it's a fully-fledged development platform that's a perfect complement to whatever other existing development environment you're working on.

Aurelio Reis

Yes. And these integrate as well into Visual Studio. So a lot of developers use Visual Studio for writing their code. There's a plug-in that will actually tie into each of these tools so that you don't have to reenter the information and you can have kind of a singular project that you can use to do your development with.

Well, so if we do a really quick dive into one of these tools, you can kind of understand a little bit how they work without going into too many details, but hopefully it gives you enough information to kind of understand what it looks like. And then in the future, we can cover in some more detail what you'll be able to do with that.

Let me -- this and move it over. All right. So here, we're actually looking at the main UI for Nsight Graphics, and we have an application loaded. This is actually Cyberpunk. And you can actually see the final output target right over here. And then to the left, one of the really important things you want to look at is the events list. So when you're writing code for the GPU, it's all based on these events that get submitted and you get access to all the API events directly here. And so you can either search or you can even sort by different properties.

So if you wanted to look at the most expensive calls on the GPU for instance, you can actually go on here and find out what that is and be able to directly identify where the most opportunity for optimization might be. One of the things that's really great about our tools is the ability to look at ray tracing workloads. So if we look for a ray tracing function like dispatch rays. Here we go. We can actually see there's a bunch of different ray tracing workloads happening.

Each of these is limited by a marker that actually establishes what's happening within that area of the code. And so global shadows or RTXDI, one of our technologies doing transparent reflections. All of these kinds of workloads are happening and there's all these dispatch rays that are initiating ray tracing commands or ray generation to occur. So ray trace actually happened.

And ray tracing comes with all these different kinds of Shader types that are really important, ray generation, miss shaders, hit shaders, and you can actually go and browse through all of these differentiators Shaders and then all of the additional information associated with them as well.

One of the really cool things that you can do is you can actually look at the acceleration structures associated with ray tracing as well. So acceleration structures are incredibly important for making sure that ray tracing can happen in real time. If you don't have acceleration structures, you're tracing rays across every single triangle in the world, which -- that's going to be your first issue when you get into performance.

And so the acceleration structure has really helped to make sure that you're only testing those rays against things that the rays could possibly hit. And so we can actually fly through the world here and inspect all the acceleration structures. You can double-click to go to when that might be of interest. And then you can even start looking at, let's say, the one, the acceleration structure with the most number of associated perimeters. And so you can imagine the highest triangle count object. And so if we double-click on that one, we can actually see it's this object right over here, which whether that makes sense or not is to -- developer to decide, but it can be very useful because they might say, actually, there shouldn't be an acceleration structure that complex. Maybe we could have gotten away with something much simpler.

Robbie Jensen

It's a good example of like an object and you've seen that you would never suspect to be dosing performance issues just like a little sitting on the table. Just because of the high polygon count for whatever reason, set under your nose is causing your GPU to spend a ton of time seculating rays bouncing off a bit. In reality, how much that little object affects the beauty of your scene and your user experience that will be up to you. But this is a great way to kind of categorize what is -- what in your environment is taking up so much time.

And I am really anxious, I wanted to ask on that same note, do the colors in the acceleration structure viewer, do they correspond anything? Or are they just used to differentiate different objects in the scene?

Aurelio Reis

So in the main view, they primarily used to differentiate and it's basically colored by the instance ID, but you can define that. So there's actually different ways that -- or different information that you can denote based on the colorization including with the latest versio of Nsight Graphics, you can actually look at Opacity Micro-Maps, which is a new technology that we created that allows you to use an alpha test map in order to not use as many polygons, but then still have objects that are very complex looking.

So you can imagine you have a tree with a bunch of leaves on the tree and you don't have to model every single leaf on it. You can have a texture that kind of represents that complexity with it. It's very optimal to use the Opacity Micro-Maps, and we actually show you that through the colorization as well.

The other thing that we can show you is complexity. And then we also have additional analysis modes that you can look at. And so as an example, we can actually look at overlaps. And we also have a heat map view that allows you to look at basically what the complexity would be for any given view so you can kind of estimate whether the performance impact for that would be substantial or not.

In the instance overlap here, you can actually see where there are a lot of these bounding volumes that are overlapping each other. And if you think about it from your shooting rays to the scene, you wanted to hit a few of these founding boxes as possible. If you have a lot of them overlapping, well, it's kind of needless because if you just merge them together, you'll know that it fits within that one objects bounding box anyway. So you don't really need to do that.

But as the rays traversing through the scene and all those computations happening in the GPU, you have to do those tests to determine whether you've hit something or not. So this is a great example of if you look at this object here, the very complex one. It's also an example -- let me reduce the speed. So I can -- a little slower there. There we go. It's a great example of here are a bunch of sub bounding boxes that are probably unnecessary. Those could just be merged on to the main model, and then as a ray is tracing through that scene, you wouldn't actually need to do those additional calculations.

And the colorization actually shows you the degree of overlap as well. And then on the other side of that, when you look at the instance map, same situation you can kind of see here where there's a lot of overlap occurring, and the complexity is quite high. And so you can use that to kind of gauge whether the performance could potentially be problematic for that area or not.

Robbie Jensen

That's really fascinating. So like if I'm building a ray trace game and I'm designing this environment, do I have explicit control over those bounding volumes and I set them? Or is that something that's more like sort of determined automatically by the algorithms or whatever engine I'm working in?

Aurelio Reis

So there are optimizations that do occur under the hood for the developers to do some degree of optimization. But a lot of this is specified by the developer directly. So it's kind of on you, the developer, to make sure that you're providing the bounding volumes that are as efficient as possible the acceleration structures that are efficient as possible. And so that includes making sure that you're setting the right flags as well.

Another thing that we ran to in the early days of ray tracing was some developers weren't using the correct flags or some of the acceleration structures that they're seeing. And so if you think about a ray that's traversing through the world, if an object is completely opaque, then the ray can just stuck, that makes perfect sense. It doesn't need to keep going. If it's translucent, that ray needs to keep going, and you'll determine the final output color based on all the additional intersections that are occurring after that, well, some developers have left the default value, which is translucent. And so the ray never stopped, it just kept going through the entire scene, which for obvious reasons, would create performance issues.

And so this is where it helps to have a tool like this where you can immediately see what these values are. And then we even created a colorization mode where we can show whether the flag was actually set properly or not, and you can specify some of this stuff on the right over here. So opaque versus non-opaque, for instance. You can actually toggle those values to highlight things that are opaque -- specified as opaque or not and really quickly be able to identify where maybe things were configured properly.

Robbie Jensen

Super cool. So this is like -- these acceleration structures and these bounding volumes, this isn't something that like the engine that I'm working in would call out to me as inefficient or something. This is something where if I'm building a game and I feel like RT is just really consuming more GPU power than it should. I would like have to look at the acceleration structures in Nsight Graphics to examine where the inefficiencies are.

Aurelio Reis

Yes. And it's all dependent on what your environment artists and your designers are doing as well. So every game -- every level within a game potentially is going to have a different performance characteristics because somebody might be coming in there and creating a situation that is not going to be most performant for ray trace application.

And so the best designers think about the creative outlet of making something beautiful, making something functional and fun, but then also how do you make it performing. And so in some cases, that something will get lost in that amazing trifecta. And so this is where you have a tool that can kind of come in there and help out to make sure that at least the performance aspect of it is considered and helped out. And for a lot of games, yes, they don't -- getting the degree of information that we have within our tools because we have more of that interface directly to the GPU allows us to provide that better for developers.

Robbie Jensen

So Kind of timely maybe because I think we all know about the Cyberpunk path tracing mode and how that's the RT overdrive fully raters environment, so full path tracing, how cool that is. Is there any difference between like profiling path tracing through this method than ray tracing? Or is it the same thing, but just more bounding volumes and more like intersections to consider?

Aurelio Reis

That's a great point. So one can have the other, but the other can't exist without the other, if that makes sense. Path tracing it's ray tracing technology. And so ray tracing can -- the basic algorithm can accomplish a lot of different things. And so you can imagine and in inclusion was something that developers have been utilizing for many, many years now. That's something that you don't necessarily need ray tracing, but at some point, people kind of started doing it within a pixel Shader, and you have the screen space amid conclusion, which is effectively tracing rays at a screen space level. With ray tracing, with DXR and vulkan ray tracing, you can now do this much more efficiently and much more seamlessly and easy and with better quality because if you think about the fact that if you screen space, you're limited by whatever is available on the screen and sometimes you'll see those haloing artifacts, they are very ugly and distracting.

You don't get that with ray tracing because it's really tracing a ray to the extent of -- beyond the screen bounds. Path tracing utilizes ray tracing by allowing you to effectively bounce these rays throughout the scene and then accumulating those results so you can get global illumination.

And so from a lighting perspective, you get something that's much more accurate because you're taking into account secondary bounces. And so without getting too detailed, you can imagine there's a light photon that comes from the sun, and will hit the ground. And it doesn't just stop. That's direct lighting, right? That's how the original lighting worked very basic lighting models. And it looks kind of flat.

And so what really happens is that light, the photon will bounce depending on the surface property. Maybe it's very reflective mirror, new surface, like water, maybe it's concrete or asphalt. And even then, there's speculate reflection that you can expect. So the ray will bounce. Maybe it will go up in this example here, it will hit this person's shirt and maybe they're wearing a bright red shirt and that shirt will accumulate some of that color. The bounce from that will go somewhere else. And you can imagine it will go down to the ground. It will go back up towards the sky. There's all these interactions that are occurring. But the logical thing that would happen is if this red shirt on a bright sunny day was right above the ground or let's say, bright pants, right, the ground would be a little bit red because of the reflection from the pants on to the ground. And that's where path tracing provides an additional fidelity.

It's much more realistic much more accurate to what you would expect in the real world as opposed to the old model where it's just -- you have somebody with bright red pants, I guess, on a white floor, and you wouldn't see any of that indirect lighting that would bounce off of that person. Only the direct lighting would be there.

Robbie Jensen

That's super interesting because then it's like every object in the environment when you're doing full ray tracing like that, like could effectively be emissive. Like I feel like with rasterization and traditional techniques, you specify which objects, like if you want to make a light bulb, this will emit light. But you would never say this guy's pants are going to emit light, but in the event that a really bright ray of light from the sun hits his pants, it would emit red on the ground around him. And I guess that probably when everything can be a missive like that, that's why it's so important for like good optimization.

Aurelio Reis

It's funny because you remind me of how -- with raster, a lot of people would start doing some clever tricks to try to mimic what would occur in a path trace application. And so with normal kind of raster based lighting, you'll have point lights, direct lights, directional lights. And you can imagine, if you just created a bunch of point lights, up and down the pants that it would -- you could color the ground red if they were red lights.

But it's a trick, right? And you're spending a lot of time and effort to try to create something that's much more natural with path tracing and requires substantially less work and doesn't have all these edge cases because, okay, you created a bunch of these lights on this person's pants to emulate the reflection of light into the ground. But what about shadows, right? How do you deal with shadows and all these lights. There's going to be a lot of problems that are created by trying to fake and kind of do it that way.

The same applies when you think about reflections. And again, screen space reflections, you could still see this in some games where they don't have the ray tracing mode activated. You'll see a reflection kind of fade out very unnaturally as you start looking towards -- like if you turn your head towards the ground, you won't see your own reflection because from a screen space perspective, you're not reflecting back. The data isn't there.

With ray tracing, the data is there because you're actually shooting the ray hitting the ground and then shooting it back up and hitting the model that represents the person's body. Very natural, very much easier to do some of these tasks that would have more and more difficult. And the quality is substantially higher.

Of course, performance, right? That's where you want to make sure that you're focusing and making sure that it works well. This idea that ray tracing is just slower? It can be if you don't use it effectively. But just in this example that I've given where you're looking down on the ground and maybe you just want to catch the immediate reflections in the certain environment, you can reduce the ray length so that you're not -- the ray doesn't hit the ground and then go up to the sky. Now of course, it will be much more accurate. You'll get the skies at the top of the skyscraper and the reflection, but will you really even notice that? So in a lot of cases, you could just reduce the ray length, and here's an optimization that you might find with the tool that helps you through performance with basically no image degradation and it's actually better than the alternative, which would have been the screen space solution.

Robbie Jensen

Right. And it's like the -- it's physically -- you're not faking anything. It's physically based. You're not doing the best estimate at what this could look like.

Aurelio Reis

Exactly.

Robbie Jensen

Cool. So is that kind of stuff calculate like is that Shaders? Is that getting in the realm of ray tracing Shaders? Or how do you calculate like rate travel length and stuff like that?

Aurelio Reis

Exactly. So within there, you can actually see some of this stuff. So I mentioned like the hit shaders and the miss shaders. In this example, I don't have the direct source code for the application. But normally, you have the high-level Shader code, and you can actually go in here and see exactly what the output would look like. And you can even edit the Shader in real time within the tool, which is incredibly cool for iteration.

If you think about, I don't want to have to go into the tool to find the issue and then leave the tool and then come back and see if it takes it, it would be great if everything could be fixed there. We actually have that capability. But from a performance perspective, what you want to do as well is you want to be able to see if the calculations from that Shader, the code that executes on the GPU to do the calculation of what the final end result pixel on the screen will look like, you can actually run a special profiler that can give you some of this information very intuitively.

I'll show you 2 ways you can do that. There's the Shader timing heat map, which actually gives you timing information on a per pixel level. So for our original view port, you can actually go in here and see kind of what the hot spots would be. And so this is a great way like you can just run the Shaders timing heat map initially and be like, "All right, that area over there looks like it's the most expensive part of my scene, I'm going to focus on that." So maybe it is that character or maybe it's that reflected skyscraper or what have you.

The other thing you can then do is once you've identified where the performance is problematic, there's multiple ways in which you can kind of understand why the GPU is not rendering or not being fully utilized most efficiently for that workload. And so what you can do is you can actually use something called the Shader profiler. And so a lot of times, when people think about profilers, they think about, will give me timing information. And for a Shader, that's not the most intuitive way, but it's also not the most useful way because you might say, okay, well, that instruction is taking me this amount of time to run but you're thinking very serially or linearly.

The way that GPUs actually work is they distribute as much work as they can to all these cores or these SMs that are actually executing these things.

Robbie Jensen

I understand that GPU, you have to think like the GPU. Is what they say.

Aurelio Reis

And if you extract it, it's not -- you don't have to think of it like all the thousands of SMs that are executing at the same time. You just think about, well, is my work efficiently distributed? But there's different metrics we can use to kind of gauge that. Divergence is a big factor, especially in ray tracing, where if you think about, you're shooting a ray, if you have a bunch of rays that are shot together in a tight cluster. There's a lot of coherence there. And you can process that really quickly because it's loading the stuff up into its memory and then just calculating all those pixels right away.

If you have a ray that goes this way and then you have a ray that goes that way, there's a huge amount of divergence. And so it might load up a bunch of memory for that -- the things in that direction and then it has to throw all that away and load up everything for that direction. And so divergent workloads can be quite problematic. And we have a lot of ways that you can visualize that within the Shader profiler. And so all of this stuff is represented the events, and we have hotspots that we actually call out.

And then another aspect to all of this is this concept of stalls. And so you can imagine busy highway with all of these lanes. And if you're able to get all of your cars going on all the lanes, you're going to get the most number of cars through a given area as quickly as possible. Well, what if there's a car stopped just in the middle of the road, in the middle lane, I think a lot of people can relate to that, especially here in the Bay Area. What happens is suddenly you have one less lane and you're not going to be able to get as many cars going through. And that's pretty much what happens when you get a stall on the GPU. You're waiting for something else to have to complete for a multitude of different reasons. And we will tell you those reasons. And based on that, there's a bunch of things that you can do to try to improve your code. So you would actually go and make modifications to your Shader code to improve your lane throughput to make it so that more instructions can be executed in parallel.

Robbie Jensen

So like if 1 lane has an instruction just stuck in it, that's blocking it, the GPU will like resource all of the workloads in that lane to adjacent lanes and that's what causes just like slowdowns across the GPU. It's not just like the 1 lane that has the stalling workload in it?

Aurelio Reis

Exactly. You'll do what's called serializing the workload. And so in the old days, you would have a serial pipeline and you would do one thing and then you do the next thing and then you do the next thing. The logical pipeline is a major improvement on that because what you do is you say, do all these things and then converge at some point. And then there's a final output that you're able to generate from that convergence. And if you have the one thing you're waiting for, even though 20 other things might have finished, you're still going to wait for that one thing. And it's -- there's no real easy way to address that outside of -- to remove that one hindrance in some way or the other.

And so we actually have different features that will tell you like there's columns that will show you that kind of stalls that you might occur and what you can do about it. There's also dependency graph that you can go in here, so you can actually show registered dependencies and be able to correlate back to where the stall originated.

So you might find that -- so as an example here, like this looks pretty bad, like the more is bad, basically more stalled samples, the worse. So as a -- from a percentage perspective, this is the area where the most number of instructions installed. There's actually a way to trace that back and figure out exactly where the instruction that originated that stall occurred and with the ability to trace that back, you can go and fix that original issue.

So I'll give you a more concrete example. You might have a for loop where you're running some calculation in that for loop. And what you might do is in order to get the original data that is doing that calculation, you might load a texture. And so you sample a texture well, well above where this for loop is occurring. Well, for whatever reason, you might have a stall that occurs when you're loading that original texture. And so what you need to do is find out what the reason for that is and make modifications to it.

But what if you were doing a lookup table for doing that calculation that's carrying the for loop. You might even realize, actually, I can just calculate that value directly within the Shader, and it's substantially faster than doing the texture look up because I then don't have that stall that gets introduced, which is, in a lot of ways, contrary to a lot of graphics programmers have tried to do in the past. Usually, people are like, well, just put it in the lookup texture and it will save calculations within the Shader.

But -- and in a traditional sense, that made a lot of sense in the way that the GPU is executing these instructions now, it may not actually be the fastest way. And so this lets you kind of like look at -- take a step back, reevaluate your assumptions and make sure that you're not making a mistake that you would think wouldn't be a mistake, but could actually be because of the way that your Shader is executing this workload.

Robbie Jensen

Right. So I want to actually touch back on -- we talked about it a little bit a few minutes ago, divergence and how -- if I'm understanding it right, so when we're doing like ray tracing, rays, it's easier for the GPU to calculate in sequential order rays that are traveling in a similar direction, effectively, to like maybe reduce it a little bit. And so like it or somehow in the queue of what the GPU is processing, there's a ray that's going left, right next to a ray that's going right and this causes the GPU to have to like backtrack, unload memory about what's left, load the memory about what's right and that can be a slowdown.

So when that happens, is that like a fault of -- did I do something wrong to get to this part? Or is this kind of like something like it naturally will happen as part of how GPUs calculate the rays? And how do I fix it?

Aurelio Reis

Yes. You're a horrible programmer Robbie. Don't do that anymore.

Robbie Jensen

No. Asking for a friend.

Aurelio Reis

Asking for a friend. No. It's a lack of information does not mean that you necessarily did something wrong. It's not always clear why the divergence might occur. And in a lot of cases, the end -- you're more worried about the end result and you're not necessarily thinking about the composition of that. And so developers have tried a lot of strategies more coherent. NVIDIA went and took care of that actually and came up with something to try to help with that. And so Shader execution reordering, SER. There's a whole white paper on it. I hardly recommend reading it. It's actually

incredibly well written and explains a lot of these concepts really well. But the idea is to minimize that divergence by sorting the rays and making sure that they're more coherent when they get executed.

And this is an API that is available to developers, and you can actually go and -- using the tool, you might find that you have a lot of spills and you have a lot of divergence that's occurring for a given part of your code. You can then go and reorder threads, real execution of the generation of these rays. So that you'll actually get a more coherent execution of directly there without having to go through the exercise of trying to figure out why are my rays not coherent enough? And how do I minimize that divergence directly by making too many modifications to the code?

In a lot of cases, there's not much of a developer can do to influence that. So reorder threads is a -- and SER is a huge, huge benefit. And the tools are very much there to help developers identify the places where they should be doing that.

Robbie Jensen

So it's kind of important that you implement SER in the right places, like in places where it will be most effective or where threads really need that extra care from SER to make sure that they're going to be handled by the GPU properly.

Aurelio Reis

That's right. Yes, there's a cost associated with that going through the reordering. And so if you don't do it in the right place, then you've paid for something that has no benefit effectively.

Robbie Jensen

Right. Right. Well, so can I say graphics like help me find the best places to put -- to use SER because I would hate to implement it, just to put it in a place where it's not going to do too much and all of a sudden, I have an overhead, I don't even need.

Aurelio Reis

Absolutely. So there's actually a bunch of additional columns that you can look at that gives you this information, including the thread instructions executed, which is equivalent to that kind of information. So you can see where that the value might be high and the places in which you might want to actually go and set up the reorder to happen directly.

Now without the direct high-level Shader code here, it's hard to determine if this is really the right place or not. There's still -- you still want to use the best judgment from the graphics programmer that created the workload because they might know that it won't apply in all cases. Again, for every level, where things can be substantially different. You don't want to just tune your render for that one specific instance, but it at least informs you to let you know if that is a place where you might have the opportunity to improve performance either for that situation or across the board.

Robbie Jensen

Super cool.

Zach Lo

I really enjoyed listening to you guys and learned a lot actually into connecting some of the dots like it's -- let me just be an artist in the past, like, this is useful material and stuff that I think a lot of people maybe they don't consider when they're building their games at times, especially when you're finding like a mesh with so many polygons, and you figure, oh, it's got 3 like separate bounding boxes pieces. So when the rays are hitting,

it's actually hitting not just one object, but like 3 different separate pieces and reflecting often how can that really help in seeing it in like visually really, really helps. And I think some -- a lot of people probably get a lot -- hopefully, a lot out of this.

Aurelio Reis

Yes. I think the challenge is -- I appreciate that. Yes. Thank you. I am glad...

Zach Lo

I like the SER explanation too. That was probably one of the more straightforward ones I've heard. Probably the -- one of the best. I found -- it's very simple to understand, right? The way you explained it.

Aurelio Reis

I find it -- yes, people have kind of overcomplicated a little bit. And that was the other thing that I want to make sure that we can cover more in the future is demystifying a few things that are perhaps more complex than they need to be. This UI is very overwhelming. And what I want to find opportunities to do is kind of like focusing on some specific bits of it and then just put the blinders on and don't let people get so overwhelmed that they're like, I have to -- I just quit. I can't do it.

Zach Lo

I think that's definitely something we want to probably cover it, like it is overwhelming when you look at it, especially the UI and you really say where do I start? But once you know what your goal is and how to get past the hurdle that you're trying to kind of hone in on, then it's a lot less intimidating, I think.

Robbie Jensen

Well, Aurelio, that was a really fun tour of -- I think a lot of the -- most of the Nsight Graphics are tracing specific features. I thought it would be kind of fun to just end up ended up -- ended off on the GPU Trace.

GPU Trace here, it's kind of, I think, I've always thought was similar to Nsight Systems, trace. It pretty much just shows all of the throughput on a unified timeline, so you can see how things correlate to each other. So the Y-axis here is time. I think I mentioned at the beginning, like what you would do is basically you attach Nsight Graphics to an executable of your game, play it for however long and then it will spit out this, which shows from top to bottom of your GPU, how the hardware units are handling workloads, what the graphics API is doing, that kind of stuff.

This example actually is a kind of interesting one showing the dispatch rays call, which is like the -- in simple terms, it's like the parent of all the ray tracing, like all the commands that's in that ray tracing, being really slow. And we actually have a video, if you want to check it out on YouTube working through how we resolved a slow dispatch rays call in Cyberpunk 2077.

But this is -- this page overall will show you how all those new great details that we were kind of looking at bubble up to your applications overall performance. Hit rays, throughput, SM occupancy, all those vocab words that maybe you don't know right now, but if you use these tools, you'll become more familiar with.

So again, Aurelio, thank you. This was an awesome tour through when like to profile ray tracing and anti-graphics. If you had any other comments on the tools or tips on how to use them before we close out here?

Aurelio Reis

Yes. I mean get them on the website and the best thing to do is play around with them. And I should mention there's a bunch of samples that we ship with Nsight Graphics. So you can just load these up and start playing around right away. These actually are part of the NBPro, samples that are available on GitHub, also created by a bunch of NVIDIA DevTechs, who are quite familiar with optimization and best practices. So definitely check those out if you're also trying to pick up on some of the best ways to approach ray tracing or just graphics development in general.

And then as we're using the tool, we make software, software sometimes works, sometimes it doesn't. And we love to hear from developers either when it does or when it doesn't. And so at the very top right, there's a little talk box. You can just click on that and actually send us feedback so that we can hopefully get back to you and improve the tool and make it even better for you. So feature suggestions or bug reports, always welcome. So feel free to reach out to us and let us know.

Robbie Jensen

Super cool. And we also plan on doing more level up with Nvidia webinars on and site developer tools in the future, and not just graphics, we can talk about Nsight Systems, which is the full system trace. Aftermath, SDK, which is for figuring out why your GPU crash, which is like the most frustrating thing ever. But Aftermath STK lives very closely to graphics, those 2 tools work hand in hand. So I think that'd be a cool one.

But if there's anything that you as a viewer want to learn about or dive deeper on, please, we've got the Q&A section coming up. Let us know what you are most interested in to learn about alongside any other Nsight Graphics questions you may have at surface during the course of the webinar.

Zach Lo

Yes, that was great, Robbie. I can be silent now. You did my job for me. Thank you, Aurelio and Robbie. It was a great session. I personally got a lot out of it. I hope some of you in the audience did as well. Like Robbie mentioned, we're planning to have follow-up episodes on Nsight scattered throughout the rest of the year. So please stay tuned and check out our socials. We do have links that should be somewhere on either side, I believe, on maybe this side here with links to pages of our dev zone all the way to, I think, forms for Nsight. So like Aurelio said, aside from in the interface, you can actually have the talk back option. You can actually go through the forms through those links to ask for NVIDIA help if you need it. And I just want to say thank you again, Aurelio. Thank you, Robbie, for coming this month. And showing everything off.

Robbie Jensen

My pleasure. Thank you for having me.

Aurelio Reis

Well be (inaudible), Zach.

Zach Lo

Yes. We will probably be having a few more episodes of Nsight. So stay tuned.

Aurelio Reis

Invite us back, please.

Zach Lo

Yes. We have our usual back to our Unreal Engine episode. We'll have with Richard next month and subsequent months as well. A lot of exciting stuff coming, and stay tuned for all that. But as for now, please stick around in about a minute or so. We're going to be here for our live Q&A. So if you have any questions right now that you just thought of. Put them in your backlog and put them in the chat because we will be here answering them live or through text.

Thanks for joining, and stay tuned.

QUESTIONS AND ANSWERS

Zach Lo

I hope you guys enjoyed. I'm a bit of a scramble so we weren't able to host any events, but now we're back. And thank you for your questions coming in. We've curated a few of them. And I think I'll let Robbie go ahead and ask -- we'll turns, shifting back and forth.

Robbie Jensen

Okay. And actually, I should introduce, we have Jeff Kiel here. He is an expert in all things Nsight, particularly Aftermath, which is fitting for our first question. So how does Nsight Graphics work alongside Nsight Aftermath. Could you talk more about how Aftermath works? How do I attach it to my project and view the results?

Jeffrey D. Fisher - *NVIDIA Corporation - EVP*

Thanks, Robbie. Actually, there's a lot of inter between Aftermath and Nsight Graphics. Nsight Graphics is really the visual component of Aftermath. It allows you to take one of those dump files and visualize all the data that we've got in there from just general GPU state as well as being able to see exactly where in your Shaders a particular problem may have occurred.

Most recently with 2023.1, we just added the ability to actually visualize some register information as well. So if you're on our Pro SKU, if you have the Pro version of it, you can actually visualize the register information to give you even a little bit more contextual data for exactly what was happening on the GPU when the exception occurred.

In terms of how to interrupt with your application, there's really kind of 2 main ways. The first way is -- and this is nice and easy because it's like 0 modifications for you is all you have to do is run our Nsight monitor -- I'd say, Aftermath monitor, that actually comes in the Nsight Graphics package. Run that monitor and add your application to the list of applications that it's going to be checking on. And when an exception happens in your application, it will automatically go ahead and dump the file out to disc. You can actually specify exactly where that's going to happen. And again, you can open those in Nsight Graphics to visualize the data.

The other way you can do it is via the Nsight Aftermath SDK. That's both delivered in Nsight Graphics kind of the most recent version will be in Nsight Graphics. But then also as we do updates if it's not necessarily at the same time of Nsight Graphics, go ahead and search for it. And again, we've had and shipped 2023.1 which came out after the Nsight Graphics delivery.

And what you do is you just integrate that, it's about, I think, 4 or maybe 5 APIs that you have to call. One just kind of initializes Nsight Aftermath, gets things set up and get you kind of connected to the driver. You also give it a number of different callbacks that we'll call at different points, including when Shaders are compiled, when an actual exception happens, and it gives the ability to specify some metadata from your application so you can even do things like at which level you were on or maybe some important information about what was happening in the scene at the moment of the exception.

And then a final call back, which is basically, okay, all the binary dump file is available, then you could either save that to disc or push it to the cloud or whatever you want to do.

The other thing to note is there is an API that was just introduced with 2022.2, I believe it was, where you basically need to ping Aftermath to find out how far it is in terms of its completion. A lot of folks, their applications will just automatically exit as soon as they see that lost device. That's obviously bad for an application that -- when you've got a background for this trying to collect a bunch of data from the GPU and from the driver and kind of push that into your application.

So you want to basically kind of ping Aftermath to say, "Hey, is it done? Have you provided all the data that you want to provide." And just make sure you leave your application alive during that time frame. And so that's your lead at it. It's just a few APIs that you need to call to be able to set it up. And then you're in control. You can just go ahead and push things to the cloud and kind of collect them at your HQ and go ahead and do the analysis there. Or if maybe you're in a QA scenario or just obviously in the development scenario, just saving right to disk and you can go ahead and do your analysis.

Robbie Jensen

Awesome. Thanks, Jeff. Zach, do you want to stir up the next one?

Zach Lo

Yes, sure. Actually, we have a -- just a new question that came into the chat, something that I think maybe we can move to the top of our list. So the question is, is Nsight -- in Nsight Trace there is a row called concurrent GPU activity that shows an event that disturbs rendering approximately every 16 milliseconds. What exactly is causing this? And is this something I can do to avoid this issue? I think they're on Windows 10.

Aurelio Reis

Yes. So there's a lot that can happen while you're taking a trace on Windows. And what we urge developers to do is basically close everything and have kind of a sterile white box environment. And so that means no Chrome running in the background. Definitely no desktop managers or other applications that you might be running that are using a graphics device. And so we'd have to look at that trace directly to understand a little bit better and kind of see if we can investigate what might be happening there. And that's something we're actually would be great if they could share that with us so that we can have a look at it.

It's something that they could submit either by sending an e-mail to us directly. And I think we do have an e-mail that they can send. I don't know if that's available here, but if they just send that to nsightgraphics@nvidia.com or devtools@nvidia.com, we can discuss it and figure out what's going on. But the concurrent GP activity usually means that there's something else happening in the background that's interfering with the collection of the trace information.

And so the way that trace works and the metrics collection works in GPU Trace in Nsight Graphics is it's actually a device level. So it happens for everything that's happening on the GPU regardless of the application that you're actually profiling. And so this is actually very useful because you can see in a lot of cases, if there is GPU contention and if there's something else that is potentially interfering with the results. And so being able to see that's helpful because there's something else that's happening there.

So I can't speak exactly to what that other GPU application or event that is occurring, it could even be something that's happening at the Windows level. I think we'd want to look at it possibly with some other tools as well. Nsight Systems is great for that. I highly recommend doing a trace with Nsight Systems to see if you can correlate that to something that might be happening either algorithmically within your application or interacting from another application. But yes, that's something that you want to investigate a little bit more just to see if there's something else that can be interfering with that collection.

Zach Lo

Great. Thank you for your question. So Aurelio referenced some material. There should be some links, I think, somewhere on either side of us. I can't remember where it is, but there is some form links that you can send questions to us, and we'll try to have someone on our team get back to you. So Robbie, do you want to take the next one?

Robbie Jensen

Sure. Yes. All right. Let's see. Okay. There's a ton of information when looking at the Nsight Graphics user interface, especially the GPU Trace, which we covered at the end there. Do you have any tips on how to make sense of or parse all of the profiler data?

Aurelio Reis

Right. So I think a good way to think about GPU Trace. It's a deep low-level diagnostic tool of the GPU, right? You get a bunch of information that's very low level. Most of that data is actionable in some meaningful way, but you have to do a lot to interpret it and understand exactly how to use that. And so what we wanted to do is make it a bit more accessible a bit easier.

We came up with this idea. Some people call it like a DevTech in a box. And so DevTechs are these experts that we have at NVIDIA that usually work directly with developers or they'll use internal tools to identify issues and a developer application and then provide suggestions. And what we came up with is a similar feature in the tool itself, it's called trace analysis. And so in anti-graphics, when you've done the GPU Trace, there's actually a button at the very top left where you can just click on it and it does a trace analysis. And this analysis engine actually goes and interprets all the metrics based or using a rule-based system that was developed by those same DevTechs that would be working directly on that application, if they were trying to identify some performance issues. And so these rules are based on best practices that have been established. So API usage patterns or specific metrics that set off red flags.

And so after clicking on that, it will actually do this full analysis and then we'll have hotspots that you can click on in order to see, which things you should focus on as part of your profiling and performance analysis triage. And so it's great because instead of having just a tremendous amount of information that's all just split out and you have to make connections and interpret, look at multiple rows and then pull up the metric view on the right and try to figure out if you're occupying, if you have high occupancy or not within the GPU workload that you're looking at. We'll basically just find areas that you should focus on and then just kind of point you in that direction.

And so really, the first thing you should do if you put on a GPU Trace is you do the trace analysis because we'll give you suggestions will point you in hopefully the right direction. And then given your context of knowing your application well and having a clear idea of what you should be focusing on, you can hopefully use that to kind of guide you in the right place. And then once you've gotten through that thing, make the modifications, go do another trace, run trace analysis again and see if you move the needle in the direction that you want. So if there was a workload that was not performing, and we gave you a bunch of suggestions, go try to implement the suggestions that are provided. And then if you see improvement, great. If you don't, iterate, try again and see if other suggestions would work for you.

Robbie Jensen

Very cool. Yes, trace analysis is such a great feature. It condenses the information overload that you can sometimes get when using Nsight Tools into just very readable, actionable information. So thanks, Aurelio. Zach, that's the -- over to you.

Zach Lo

There is actually a follow-up on the question that was asked previously about the Nsight Trace. Is this using Nsight Systems or Nsight Graphics GPU Trace?

Aurelio Reis

It's a really good question because they're both very similar in that regard. There's the trace functional in GPU Trace and then there's Nsight Systems, which is a trace-based profiler, which provides very similar metrics collection. And I'll tell you, it's actually using the same back end. We try to share as much as we can there.

GPU Trace is more graphics oriented. And in the future, we're going to have more connectivity between the kinds of graphics concerns that you might have. So being able to look at resources and tie it into the Shader profiler and being able to edit shaders and then run another trace and see if you've been able to make progress in what you're trying to achieve.

Systems is more systems-oriented systems level where you're looking at everything that's happening across the PCI bus. You're looking at the algorithms that might be running on the CPU and then submission of that workload to the GPU and trying to find inefficiencies in the way in which the CPU is communicating to the GPU. You can starve the GPU by not feeding it enough information. They're incredibly powerful, and you can feed it a lot. But this is something where if you're not -- if your algorithm is not conducive towards submitting that information fast enough, then you're basically -- you have a very fast expensive GPU that is majorly underutilized, which is a horrible waste, especially for your users.

And so you kind of want to do both. I mentioned it earlier in the video section, but the main thing is you want to start with Nsight Systems, validate whether you're actually feeding the GPU effectively. At some point, you are going to reach a place where you're probably saturating the GPU, you're trying to submit more than it can reasonably run through. And that's something where GPU Trace would be -- more of that information to help you out there.

Jeffrey D. Fisher - *NVIDIA Corporation - EVP*

I do want to emphasize one thing on the trace analysis. One of the things that people don't understand is how powerful that is. It's actually the -- it's actually based on the exact same code that's used by our developer technology group, which are the ninjas, the gurus whatever you want to call them, that go in and do the in-depth analysis of all these high-end engines and give them feedback in terms of what they need to do to optimize their performance. And so rather than you having to like make a phone call or whatever, we basically give you the exact tools that they're using. So the information is like super, super tight.

Zach Lo

Great. Thanks, Jeff, for pointing that out. All right. Moving on to the next question. This is about making a game in an engine. So I'm making a game in Unreal Engine. Do I have to build it as an executable to run Nsight Graphics on it? Or can Nsight profile it inside of Unreal?

Aurelio Reis

Right. So Unreal Engine does a lot of what some people call baking, right? It has to do kind of a post process on the information that is generated to render the scene efficiently. And so in the editor, it's very different than when you're actually running the application out in the wild. And so there's -- it's a challenging situation for tools because, well, if you're running into a performance issue in the editor, well, maybe it will help Epic go and optimize their editor to be faster, but it's not necessarily helping the developer because it's not the optimal fast path. And so the recommended approach as of today is export the application so it has these optimizations in place, and then run the tools on that version of it.

Now we recognize that slower than you might want because you want fast iteration in Unreal Engine. It's one of the reasons probably why you use it, same as other engines like Unity -- and so one of the things we've talked about and that we're investigating is can we integrate the tools directly within the editor itself so that you can just save, capture the frame or do a trace directly from the editor window and have some actionable information.

Now again, it won't be 100% accurate because all of these optimizations that the engine does did not get to happen. But at the same time, hopefully, you would get some indicators that help point you in the right direction. So if your performance was 20%, 30% over what you expected, or under what you expect, you probably have enough room where the tools could be useful. If it was 5%, 10%, I mean that's the kind of -- that's in the noise, right? That's where the engine does enough optimizations to eliminate those kinds of inefficiencies.

Robbie Jensen

Great. Thanks for answering all these questions. So we're running low on time, and I think we have time for one more question. Okay. Okay. Let's see here. This will be our grand finale. Okay. Can I use Nsight Graphics to profile accelerated parts of my game? For example, I'm trying to optimize particle systems. They cause a lot of lag, especially when the camera gets up close to them. Can -- I can figure Nsight to examine just what's going on with the particles?

Aurelio Reis

That's a great question. One of the things that you want to do. So it's usually the way that we even think about analyzing this kind of -- these kinds of issues when we're looking at applications that are nonperforming, we think of the workloads that they're submitting. And if you think about it from a compute perspective, you usually do like a dispatch. If you're doing ray tracing, you're doing a dispatch rays. When you're doing graphics, well, you're doing a lot of draw commands. And so there's not like that atomic individual way to establish here's all the work that's being done. And so the way that developers should be representing that is by using markers. And so PIX Markers for D3D, that's a very popular approach that most developers are using.

There's Vulkan Markers equivalent as well that developers should use for Vulkan. But we actually recommend there's our own library that allows you to get this information through something called Vpx, and there's even more contextual information that you can provide using these kinds of markers.

Now it's up to you, whatever you decide to use. But the main thing is, as long as you're using these markers in the right place, you can establish the kind of workload that's happening. And then, for instance, in GPU Trace, going back to that, when you do a trace analysis, we actually use the markers to make the determination on where the optimization should occur. So if you have a marker region that was called render characters, then you may see all the metrics associated with that region and then the potential improvements that you could make for your character rendering. So maybe it's something with the subservice scattering of their skin or their weapon Shaders are too expensive.

I don't know how you do your characters, but that's the thing where you can also get more granular. So you might have a character and then you have vehicles and you have weapons and you have whatever other atomic primitives that you want to start tracking that way.

So that's in general for rendering or crashes you also can do Nsight Aftermath markers. And this is actually incredibly useful because if you have a GP crash, let's say, that root signature wouldn't set it properly. And so there's an exception because something is pointing to now. You can usually get enough correlation formation from Aftermath, but in some cases, it's hard to find out exactly where in your code that relates to. If you put down these Aftermath markers, you can see exactly where the crash occurred, and be able to debug it much, much faster.

These kinds of crashes, we hear from developers take a lot of development time to find. And so this is a huge, huge benefit, especially because markers, there's actually an update that's going out where they're going to get even faster. And so then you're going to be able to enable those for players out in the wild. And so if you have your QA team with, I don't know, 20 people that are filing bugs for crashes that are occurring internally, but now you have 100,000 users or 1 million users or 10 million users that are going to see those crashes much more than those 20 people. You now have all this correlation information and these crash dumps that provide much more actionable data. And so instead of having a bad customer experience, you have something that's going to work way better because you're able to identify the issue and fix it much quicker.

Zach Lo

Thanks for the in-depth response. Well, guys, that's all the time we have for this episode. I want to thank Aurelio and Jeff and Robbie for joining us this month. And thank you all for -- out there in the audience joining us and asking your questions. We value your time and joining us. Please keep in touch with all the links that we have in the window, ask questions, and we will see you next. So thank you, guys, and thank you all for joining.

Robbie Jensen

Thank you very much, Zach. Thanks, everyone, for watching.

Aurelio Reis

Bye.