CG GARAGE PODCAST #271
ALEXANDER SOKLEV
V-RAY GPU TEAM LEAD, CHAOS GROUP

V-Ray GPU is awesomely powerful — and with out-of-core rendering, it's only going to get better. Join Chaos Group's Alex Soklev for a peek behind-the-scenes.

Over the past 12 years, V-Ray GPU has been developed alongside the CPU renderer to take advantage of increasingly powerful and specialized hardware — and now it's coming of age. Joining Chris this week is Alex Soklev, whose passion for ray tracing has propelled him to the position of Team Leader in Chaos Group's V-Ray GPU team.

Contents

Useful links

Alex's GTC presentation (requires registration) >

V-Ray GPU at Chaos Group >

V-Ray GPU/NVIDIA RTX support blog post >

Chris Nichols	You haven't been on before, which is actually cool to have you on for the first time.
Alex Soklev	No, no. Yeah, I haven't been on.
Chris Nichols	But-
Alex Soklev	This the first time, so I'm pretty excited.
Chris Nichols	Yeah. So, it's cool because you've been doing GPU stuff for, how long have you been on the GPU team now?
Alex Soklev	Well, in the beginning when I started at Chaos there was a small timeframe in which I wasn't sure which team I was in. But, back then I think the differentiation was not that good. So, there were no specific teams, just people were helping each other and trying to do their best to make everything work. Then team started to-
Chris Nichols	Formalize?
Alex Soklev	Yeah, formalize here and there. And I think I'd been on the GPU team as long as it's been there, like day one, I guess.
Chris Nichols	Yes. So, that's at least what, six years?
Alex Soklev	Five years plus something. Yeah, almost six years. I've been at Chaos for almost six years now. Next month is my sixth year. Okay.
Chris Nichols	Yeah, I just had my six years, so you were just after me.
Alex Soklev	Yeah. When did you start?
Chris Nichols	November 2013-
Alex Soklev	'13-
Chris Nichols	... I guess.
Alex Soklev	Yeah.
Chris Nichols	Yeah.
Alex Soklev	So, I started June, beginning of June, 2014.

Chris Nichols Right. Okay. cool. So, anyway, so you were ... it wasn't formal that they had the GPU team, but it is now, and you're now the head of the GPU team-

Alex Soklev Yeah.

Chris Nichols ... which is a big responsibility. And it's been pretty cool. Let's give people a little bit of a story of how did you get into programming, how did you end up at Chaos Group? How did that happen?

Alex Soklev Well, that's a funny story. So, I guess I started programming because I was good at math, and I was ... after I graduated high school, I was thinking, "What should I do with my life, and how should I approach stuff, I need to get a job." And since I was always good at math, I decided, "Well, programming is good for me."

Alex Soklev I never did programming in high school. I came from a language background, high school. No deep involvement in mathematics or programming or anything like that, but decided to go for it, and started studying computer science at the University of Sofia. And I studied like two years, and at the end of the second year, I didn't like it at all. I thought this was not the thing for me. So, I just quit and decided I'll go pursue something different.

Alex Soklev I was really into 3D. Ever since high school, I was playing around with Max and Maya. During the year when I was out of university, I was following tutorials, and because my dad actually is in the movie industry, he's an electrician.

Chris Nichols Oh, nice.

Alex Soklev I don't know exactly what the term is, I think it's gaffer.

Chris Nichols Yes. But, he basically was in the lighting team.

Alex Soklev Yeah, the lighting team. Yeah.

Chris Nichols Yeah.

Alex Soklev So, he met me with some guys who are doing CG here in Bulgaria. So, I started following them and trying to learn here and there. And while this was all happening, some friend of mine from my class at the university told me that there's this guy that's holding a brand new course at the university called ray tracing. And they're doing ray tracing. And based on what I've told him, like, what my interests are and what I like he told me, "You should go there and check this out."

Alex Soklev So, I went back to university. You could attend this even though you weren't enrolled this year. And it was actually the best thing that I've seen so far about programming. And I suddenly decided I want to be ... this is what I want to be, this is what I want to do. So, I-

Chris Nichols So, suddenly you married your 3D interests. Your 3D interests and your computer graphics interest suddenly found a way together.

Alex Soklev Yeah. And, that's what brought me back to the university. So, I enrolled back next year, and I graduated.

Chris Nichols Wow, there you go.

Alex Soklev Just when I got back in the university, I had an ... in the year that I skipped, I didn't do any programming at all. And I was really, really bad at programming back then. Then I decided to apply to Chaos. I think it was 2011. I went on an interview, and I sucked a lot. Now when I remember how the interview went, I'm even amazed that they actually invited me for a second interview a few years down the road when I applied the second time.

Chris Nichols Wow.

Alex Soklev But I guess the second time's a charm, so, yeah.

Chris Nichols Right. Wow. Okay. All right. So, that's how you ended up ... your interest in computer graphics led you to ray tracing and programming. So, that was pretty cool.

Alex Soklev Yeah. And sheer luck. You know Vasko, unfortunately, he's no longer on the team. He's the guy that held the ray tracing course at the university, and he's the guy that sparked the light in me.

Chris Nichols Okay. Well, that's fascinating. And that's good because I think what we're going to try to do in this podcast, is we're going to have a refresher course on what ray tracing really means, especially for people that can say, talking about GPU ray tracing, I think that there needs to be a better definition to define here and all the different problems and issues that are going on there. Obviously, the big thing now is GPU ray tracing, right?

Alex Soklev Yeah.

Chris Nichols So, a lot of people are talking about its capabilities, et cetera. And Chaos group has been working on GPU ray tracing for 12 years, 2008 is when we first started looking at GPU ray tracing. And back then I don't think it was even ... yeah, it wasn't even with CUDA. I think it was trying to shove it into a shader somehow, or something weird that we were trying to do. Our very first implementation was not even through CUDA.

Alex Soklev Yeah. The very first implementation was actually in OpenCL. It was the first API for general-purpose programming that came out. It was the first thing that allowed you to do general-purpose computing on the GPU. Before that, it was hacked through shaders. And as soon as OpenCL was out, we started doing ray tracing on the GPU. And I think it was about a year later that CUDA was released. So, we naturally brought the two of them together, and we started building a unified ray tracer that would work on both.

Chris Nichols Right. Well, that's actually interesting. So, we should do a little refresher on what an API is because that actually ... This is the way we started. We actually did have two different APIs going for a while. We had the OpenCL version and we had the CUDA version. And, the idea was that with OpenCL, we were just basically keeping our options open because, depending on which hardware people wanted to use, it wasn't necessarily tied to a specific piece of hardware, because CUDA is tied to the NVIDIA hardware. And the OpenCL API is tied to any hardware including CPUs. You can use CPUs as well, right?

Alex Soklev Yeah.

Chris Nichols So, that was a big choice. And at some point we abandoned OpenCL, and it took a long time. We kept it going for a long time. But we abandoned the OpenCL because, I guess, we just weren't getting the performance or the features that we were getting from CUDA, is that right?

Alex Soklev Well, sadly there's a ... OpenCL as an idea was perfect. So, it was a portable API, you could write your programming in OpenCL and it could run everywhere on any GPU from any maker, even on CPU. It was great in theory. Sadly, in practice, if you want to be able to attend to everybody's needs, you need to make sacrifices. Like, you need to cut this corner and that corner in order to suffice everybody's needs.

Alex Soklev And what eventually OpenCL became is a less powerful API than CUDA. CUDA was very tied to the Nvidia platform. It was very performance, very optimized on the Nvidia hardware. We were trying our best to keep them both together. But at one point it just became too much of an effort on our side to maintain OpenCL. There were just too many features missing that we needed in order to give our clients the performance that we know we can extract from the GPU. And, this is eventually what led to our abandoning of the OpenCL platform. Also, there was-

Chris Nichols Yeah, go ahead. I mean, it's also the fact that no one was ... we knew how many people were using NVIDIA hardware versus using alternatives, right?

Alex Soklev Yeah.

Chris Nichols So, there were very few people, I mean, almost every one of our users was using NVIDIA. So, there was really no reason why we shouldn't just stick with CUDA the whole time because that's 99.9% of the time, that's what people were using, was NVIDIA hardware, right?

Alex Soklev Yep, that's true.

Chris Nichols So, yeah, supporting an API that basically would only satisfy 0.1% of our user base isn't necessarily a good idea.

Alex Soklev As long as it was easy, we didn't mind it, but it was getting harder and harder all the time.

Chris Nichols So, for a while we were sticking with one API, and then suddenly now we have another API.

Alex Soklev I guess that's the work of the GPU developers. There's always another API just around the corner. Just when you think it's getting easier, no, it's not.

Chris Nichols Yeah. So, we still have a very robust CUDA base, right?

Alex Soklev Yeah.

Chris Nichols Right now CUDA is extremely good, it's very feature-rich, and it works really well, and it's really fast. And, we've added a second one, which is the Optix one, right?

Alex Soklev Yes, this is true.

Why we added Optix

Chris Nichols So, let's explain why we added Optix.

Alex Soklev There's a very, very simple reason actually why we added Optix. As you know, the last generation of the NVIDIA GPUs, the Turing architecture, brought a very cool new feature to everybody. It's the RT core. So, the RT core is a piece of hardware that's inside your GPU, and it's actually a separate processor designed specifically for ray tracing.

Alex Soklev So, it can do ray-to-triangle intersections, or it can traverse trees for you. And because it's done in hardware, it's much, much faster than anything you can ever do with software. And the only way for us to access this new goody was through the Optix API. There is no direct access to the RT cores through CUDA. So, if you want to benefit from that speed up that you'll get if you do your intersections in hardware, then you need to use the Optix API.

Alex Soklev	And this is why we started writing and rewriting V-Ray GPU over the Optix API. And this is something that we've been doing for quite a while now. Actually, the first time we heard that this is going to happen, Optix was version 5.0, I think. So, we started doing this with Optix 5.0, which is I think more than two years ago, three maybe, years ago.
Alex Soklev	Through the course of time Optix evolved a lot because this is something that nobody has ever done before. This is a brand new thing, and nobody was sure what's the proper way to do it. So, when we first brought it in Optix 5.0 it wasn't perfect yet. So, there was a lot of back and forth. We were very closely partnering with NVIDIA and sharing our feedback on what we need, what the API would need, and how it needs to evolve and what needs to change in order to be really production-ready for everybody out there to use in powerful production software like V-Ray GPU.
Alex Soklev	And, we went through a lot of back and forth. There was Optix 5.0, 5.1 5.2, then Optix 6.0 came in, 6.1, 6.5. And eventually, we arrived at Optix 7.0, which is the final version that we are now running on, and we are feature-complete, just as CUDA is. So, every feature that we have in CUDA is there in Optix as well right now.
Chris Nichols	And that is a ... we should note, that that is a huge effort, like to try-
Alex Soklev	Yeah.
Chris Nichols	... to make everything work from one API to the other. That's like basically rewriting ... designing an entire new car to make it look exactly the same but from scratch.
Alex Soklev	Yeah. So, we basically changed the complete backbone of the render. We had to revise everything from scratch. And something that we're really proud of is that we were able to design this and make the whole effort in such a way that currently, we're using more than 99% of the CUDA codebase for Optix.
Chris Nichols	Interesting.
Alex Soklev	So, we changed our CUDA code in such a way that ... so, it shares the entire codebase with Optix right now. So, this is something that would-
Chris Nichols	So, you changed both codes?
Alex Soklev	Yeah, we had to change both codes.
Chris Nichols	You changed both codes, yeah. Wow.

Alex Soklev But, we adjusted the CUDA code so it fits Optix in such a way that now they share the whole codebase. So, in order to support the two of the APIs together, we actually, it's a normal operation for us. It's easy. Every change you do in CUDA comes automatically in Optix. And the other great benefit of this is that no matter what you do, you know that the results you'll get from Optix are exactly the same as the results you'll get from CUDA.

Chris Nichols Wow. There you go. That's a big deal. That's a really big deal.

Alex Soklev Yeah. It tells people that they can switch from CUDA to Optix at any point, and it will be safe.

Chris Nichols So, we should note that when we go basically ... if you go into your current version of V-Ray in Maya or Max or whatever, our current released version of V-Ray does actually have the Optix option in there, right?

Alex Soklev Most of them do, and if you're on a DCC or a platform that still hasn't gotten the latest update, which is internally, it's version 4.3, which was for Max, it was update two, I think, for Maya it was update one. And I know that it's already shipping for Rhino, Houdini and SketchUp, but all the rest are coming later this year.

Chris Nichols So, the big thing to note is, what you'll notice is that you'll have an option to render it in CUDA or render in Optix. It is actually going to give you the same thing. The really, the only difference between those two options is what hardware you have that supports it, right?

Alex Soklev Yeah.

Chris Nichols So, if you have new hardware, like a new RTX card, then it is highly recommended, you just might as well just use the Optix version. The Optix version is what you should be using for any GPU stuff, right?

Alex Soklev Mm-hmm (affirmative).

Chris Nichols Okay. And then the one thing that the Optix version currently does not support is adding the CPU as an additional processor. Because, I think the CUDA one does, but the Optix one does not support that yet, right?

Alex Soklev Yeah. This is actually the only thing that Optix doesn't have, and there is a very good reason for this. And it's because when you render with CUDA everything is in our control. So, we build everything, the acceleration structures, the geometry, everything's in our control. And when we have these acceleration structures, we don't care who's going to use them, like, the CPU or the GPU. But when it comes to Optix, we no longer are in control of traversal because it's the RT code that does the traversal.

Alex Soklev	So, suddenly we don't have those trees, we don't have access to them, so we can't just tell the CPU to go and help the GPUs. If we want to do that and enable the CPU to do work parallel with the GPUs, we need to go through the step of building the trees on our own just for the CPU. And currently, this is a very time-consuming step. It's okay to do it when you do it for the GPUs as well, but now Optix handles that thing, and it does it blazingly fast. And, we are afraid that if you do that, it will actually maybe decrease performance.
Chris Nichols	Right. Because you basically have to write it for Optix, the tree will have to be written for Optix and then also written separately for the CPU.
Alex Soklev	Yes.
Chris Nichols	Right.
Alex Soklev	Yes. And also it would be probably two different algorithms. The Optix algorithm for a tree intersection is not open, so we need to match it somehow in the CPU in order to guarantee that you'll get the same result.
Chris Nichols	The same result, yeah.
Alex Soklev	So, this is kind of an issue. But what you can do, because the CPU and the GPU codebase are exactly the same and the CUDA and the Optix codebase are exactly the same, you can still, for example, if you have an RTX card on your own machine and you want to offload your render for final frame rendering on a CPU render farm, you can still use CUDA CPU, like, the hybrid mode for CPU only in order to exactly match your RTX result.
Chris Nichols	Right. So, basically you could do it in RTX and then switch it to CUDA and send it to the farm?
Alex Soklev	Yep.
Chris Nichols	Okay. All right. Well, that's actually-
Alex Soklev	You can do it locally with RTX and you can send it to the farm using CUDA.
Chris Nichols	Yeah. Okay. Well, that makes sense. Okay. So, let's talk a little bit. Obviously, right now we're all stuck at home doing our thing, and unfortunately, that means you had to miss the GTC conference. And you were supposed to give a talk there, and you did give a talk, but you had to give it remotely this time. I watched your talk, which was excellent. Very good talk-
Alex Soklev	Thank you.

Chris Nichols And, there were a lot of interesting questions. We should put, you know, for those of you who want to see it, I think it's free to register for the GTC conference, and then you can see Alex's talk on there. If not, it may just be online at some point. But, we'll try to find a way to make sure you guys have a link to it, and somehow check out Alex's talk because it's very good and it's very concise, which is actually very-

Alex Soklev Actually there is a link in the forum of V-Ray GPU. So, if you go to Chaos Group Forums, and you go into the GPU section, you can find the link.

Chris Nichols Perfect. But we also will put it in the show notes for this podcast as well, so people will have that as well. So, just go to the CG Garage page, and we'll put the link in there.

Alex Soklev Great.

Chris Nichols Okay. So, a couple of things. One of the big things ... well, let's first talk about ... let's get a little bit broader. Let's talk about the idea of what people are calling GPU ray tracing because I think people are being very loose with this term ray tracing on the GPU. You know what? Actually, no, let's not do that. Let's go back to V-Ray. Let's go back to V-Ray specifically.

Defining out-of-core

Chris Nichols Let's talk about out-of-core because this is a big thing that you've been dealing with specifically. You and I have been having a conversation about this. I've been trying it out, I've been testing it, I've been writing blogs about it. They haven't come out yet, but I'm still going into this. Out-of-core is a big, big deal. So, let's explain what that is, and the different levels of out-of-core, because I think that was something that was very interesting, the way you broke down to me what out-of-core means at different stages, right?

Alex Soklev Mm-hmm (affirmative).

Chris Nichols But let's first define what out-of-core is. Explain that for us.

Alex Soklev So, out-of-core, in terms of GPU, is the ability to offload some of your memory to the system memory, the CPU memory in order to make room for more stuff from the GPU. So, this is a very, very important feature for GPU rendering specifically. It is something that all the operating systems support by default. So, on the CPU you have out-of-core naturally. It comes with the system. On the GPUs though it has everything-

Chris Nichols But, it goes to the hard drive.

Alex Soklev Yeah, that's another level of out-of-core. So, it goes from the system RAM to the hard drive. And from there it can even go somewhere on the network drive where you have an infinite amount of space. But, every single hop, every single one of these means order of magnitude, slower access to that memory. So, if you're using your system RAM, for example, in the CPU, you have 12 gigabytes, 16 gigabytes of RAM, you're doing something that's taking a lot of RAM. Suddenly, you're out of RAM.

Alex Soklev There are two cases of what’s going to happen in this scenario. One, you'll have disabled out-of-core. This is an option for you. You can disable it in the system settings. And when that happens and there is no RAM on the system, your application will 99% of the time just crash. So, there's nothing to do. You need more memory, there is not enough memory, the application crashes.

Alex Soklev If you have out-of-core enabled, and this is virtual memory on a regular system, it will start offloading some of your memory onto the hard drive to make room for the new allocations that you need, the more memory that you need. The problem though is that at some point your application may need back access to the memory that it has offloaded to the hard drive, for which the computer needs to remove another chunk from the system memory to a different location on your hard drive to bring the first location back in.

Alex Soklev And this is a very tedious and very slow process, because you need to go to the hard drive, you need to synchronize, you need to read chunks of memory, you need to bring them back to the system memory. It's a slow process. So,

Chris Nichols And hard drive is much slower than memory than-

Alex Soklev Much, much slower.

Chris Nichols ... the system?

Alex Soklev Yeah. It's orders of magnitude much, much slower. Even if you're running the fastest hard drive out there, like the fastest-

Chris Nichols SSD.

Alex Soklev ... SSD on a PCI slot, it will still be an order of magnitude slower. So, you have probably seen this, when you're out of system RAM your computer just becomes extremely sluggish, and it's hard to even move your mouse around — but it doesn't crash, it works. And this is the good thing about out-of-core because for example, imagine you're running a GPU render overnight, like you're running an animation. And in your animation there's this one frame that would go out-of-core. It will need more memory than you have on your GPU.

Alex Soklev Instead of crashing at that point, out-of-core would allow you to still render that frame. It will be slower because you would need to evict some memory from the GPU via RAM to the system RAM, so you can make room for the additional stuff that you need on the GPU. But eventually it will render out. And if it's, for example, just this one frame in the morning, you can come safely to work and know that all your animation is done and ready. You pay some, probably a few extra minutes for that render of that specific frame, but the whole thing is done. And this is the cool part of out-of-core, that it will give you this safety that you know that the render will come through.

Chris Nichols Right. But it's very ... it's something we've been obviously looking at for a long time, right?

Alex Soklev Yeah.

Chris Nichols It's challenging because just the idea of adding out-of-core can significantly slow down your render, even if you don't need the extra memory, right?

Alex Soklev Yeah.

Chris Nichols Just, it's a much more complicated way of looking at things, right?

Alex Soklev Memory access is the fundamental part, probably, of every program. In this case, you need to replace it. It's a very hard problem. So, if you start with out-of-core in mind from the beginning, it's probably easy, but V-Ray is, as you said, it started 2008, V-Ray GPU. There's a lot of things in V-Ray GPU right now. It's a production-ready renderer, it's very complicated, there's tons of features.

Alex Soklev And, just going out-of-core at this stage is very, very hard, and you have to take a lot of things into consideration. And every change you make could be a potential disaster to performance. So, we're very careful in how we make out-of-core, how we make our accesses, how we group requests for a memory, for example. Everything needs to be very, very carefully selected, and very carefully prioritized in order to give the performance that you need.

Alex Soklev So, even if you don't go out-of-core because you have changed the backbone, you have changed basically everything, you can still suffer a performance penalty, and this is what we're trying to avoid here. We're trying to make it as perfect as possible, so when you don't need it, you don't pay the extra price for being able to go out-of-core when you need it.

Chris Nichols Right. Okay. So, let's talk a little bit about what is in memory. Because the most expensive parts of the memory that happen on a GPU are two things; geometry and textures.

Alex Soklev Yeah.

Chris Nichols So, there's several ways, and it was really cool the way you outlined it to me in that email that you sent me a few months back. It was like, "There are several ways to think about it. One is, on-demand memory." Where you load things slowly as you need them only at the level that you need them for that specific purpose, which means, let's say you don't have out-of-core, the thing you can do, let's say is like, I just load everything whether I need it or not. Every resolution in every piece of geometry or whatever, and then it will render, because then I just have everything in memory. But, you don't necessarily have that luxury because GPU memory is expensive, right?

Alex Soklev Yeah.

Chris Nichols So, you do something that's called on-demand loading, right?

Alex Soklev Yeah.

Chris Nichols And I think we've done it for ... we've had that for a little bit on the texture side of things, right?

Alex Soklev Yeah. We've actually had it for a few years now.

Chris Nichols So, explain that, because it's really pretty simple if you think about it. But go ahead and explain how that works on the texture side.

Alex Soklev It's a great feature to have, and it's something that we will extend to out-of-core later on. So, what on-demand textures is, as you said yourself, when you start rendering, you need access to all that data. For example, when a ray hits a surface, it needs to evaluate the color, input the color, it needs all the vectors that this specific material needs. And because in the beginning of the rendering, you have no idea what you're going to need in the scene, you just load everything. And, you loaded all the different resolutions and mipmap settings and whatnot.

Alex Soklev And this can be really, really expensive on the VRAM. People are constantly using a ton of 8K textures and all those textures, when expanded in memory, take a lot of RAM. And as you said, VRAM is precious because you can't upgrade it. You can't just put in more VRAM. What on-demand textures does is it's a mechanism that allows V-Ray to start rendering without uploading any textures to the GPU. It just uploads placeholders. So, tiny descriptions of what this texture is so that when a ray, during intersection, hits a material, it can request, from the CPU, the texture it hit.

Alex Soklev And, not only it will do a request for the specific texture, but now that you know exactly how far away you are from the camera, you know how much the ray has dispersed, and you can take a lot of extra information because you're actually at runtime, and you're exactly at this point at this hit. You'll have all the data you need so you can make a request for a specific level of the texture. So, you don't need the whole thing, you can just load a smaller piece of it, then you can filter it, or pre-filter a tile on the CPU so you can just get exactly what you need on the GPU, not the whole thing.

Alex Soklev And this happens for every ray, so essentially you only load the textures that you actually need, not all of them. And you can downscale them to exactly the resolution you need, not the full size. So, if you're looking at the texture from close-up, it will be absolutely full size, but if you look it from a really, really long distance, it will be just a small piece of filtered, blurred texture that would have absolutely no visual difference from the whole thing, but it will save you so much memory. So, I've seen scenes go down from tens of gigabytes of textures down to single gigabytes, just one, because of this texture.

Mipmapping 8K textures

Chris Nichols Well, that's the thing ... Basically what you're saying is people can be lazy and inefficient, and make everything 8K textures, and it doesn't matter anymore, right?

Alex Soklev Yeah.

Chris Nichols Because basically we'll just load a resolution ... we'll resize all your textures down to multiple levels, which are called mipmaps, right?

Alex Soklev Mm-hmm (affirmative).

Chris Nichols And we'll load the resolution that we actually need for the renders. So, you don't say, "Ooh, I need to make it 2K because I'm worried about RAM". Don't worry about it. Just make it all 8K if you want to, 16K it doesn't matter, that's all the disc space you need. But, you can still basically do that. Now, this is an important part of on-demand, because what you basically outlined to me, which is very interesting, is that it'll load it in, but it won't unload it.

Alex Soklev Yeah.

Chris Nichols That's the thing. That's the new thing.

Alex Soklev That's the difference, yeah.

Chris Nichols Okay. So, let's explain that.

Alex Soklev	So, with on-demand, you can take stuff in, upon request, but there is no mechanism to offload things. So, this is what we're actually working on right now. So, when you implement the eviction, the possibility to throw things away, now, this is the place where you go out-of-core. But, it's a hard thing to do because there's a lot of issues and problems that come with eviction. So, whenever, for example, you hit something you need, for example, a piece of geometry that you didn't have now, you need to make room on the GPU for that piece of geometry. But you need to throw something out because there is no space. The decision, what you should toss out is very important because it could dramatically affect performance. Also, you need to make sure that nobody else is currently using this. And on the GPU, this is hard because, for example, on the CPU you have like eight or 16, or even, let's say you have a very powerful Xeon processor, you'll have like 100 threads, but on GPU you have 3,000 and more, and you may have more GPUs. So, you need to make sure that nobody needs this-
Chris Nichols	So, you have to check with 3000 threads and say, "Are you sure you're not using this?"
Alex Soklev	You have to make sure that nobody needs this. You have to do this without locking everybody, because everybody needs to work, and you have no option to tell anybody, "Stop working, I need to check something." So, everything happens at runtime. So, all those threads are just humming there, rendering, and you need to be able to throw stuff in, throw stuff out, bring stuff in. And this is the really hard thing.
Alex Soklev	Because, for example, with on-demand, you know that when you hit, for example, a piece of geometry that it wasn't there, you know that it wasn't there for everybody. And when you bring it in, now it's in for everybody. And you know that it's going to stay there forever. But, if you evict it at some point, now you have to take extra precautions, because something that was there just a second ago can suddenly go missing. And you need to handle that case. This is much, much harder than it sounds, and this is why we're taking our time to make it perfect.
Chris Nichols	Yeah. Well, this is a very big challenge. I know it's a very big challenge because, obviously, this is why we've been working on it for so long. But, it is an interesting problem, because basically, when you go through the bus, when you go from one kind of memory to another kind of memory, you want to go in that area as few times as possible for a shorter time as possible, right?
Alex Soklev	Yeah.
Chris Nichols	And so, predicting when you can get in and out is the big issue, right?
Alex Soklev	Well, when you need to go out, you need to go out. So, you're going to pay the price from going through the bus from one memory to another, that's a given. But you can make this worthwhile. So, you can cram as many requests, for example,

into that single go-back-and-forth as possible in order to keep performance up. You can't simply do this transaction from one bus to the other every single time you have a request. You need to gather those, you need to issue requests at specific intervals, for example. So, you make sure that the performance is optimal so you can utilize this bus as much as you need to, otherwise you will suffer.

Alex Soklev And, another thing in out-of-core that is very, very important, and I think a lot of people when they talk about out-of-core, they're just thinking about what happens when they go out of memory. And what's the price that I'm going to pay once, for example, I have a 10 gigabyte GPU, I have 12 gigabytes of scene that needs to go in. But, something a lot of people, I think, overlook, and this is a very, very important thing, is that, are you sure that you need all those 12 gigabytes of your scene?

Alex Soklev A very important aspect of out-of-core is, for example, it would allow you to render scenes, like 100 gigabyte scenes, into your 10 gigabyte GPU. If you were using a huge scene, like a huge city, with skyscrapers and whatnot, and your camera is positioned in a single room in one of those skyscrapers, you don't need to go and simplify your scene, and delete whatever you don't need in order to render it so you can fit in memory. With out-of-core you can just render that scene straight out, and the renderer will automatically load only what it needs.

Alex Soklev So, it will start even without geometry, even without textures, just boxes and placeholders for structures. And whenever a ray hits something that it needs, it will load it into memory. But for example, in this case that I just gave, you are in a single room, so you're most probably going to need everything that's in the room, and everything that you see directly outside through the windows and nothing else. So, you won't load the whole scene, the whole city, it would only load what you can see. And this way, your scene might fit in your memory, and you will never go out-of-core in the first place.

Chris Nichols So-

Alex Soklev You will utilize only-

Chris Nichols So, but isn't that basically ... But it is on-demand then, right? Basically it's-

Alex Soklev Yeah, this is on-demand geometry loading.

Chris Nichols Okay. All right. Which is fine, which is great. So, how does on-demand geometry loading work? Because you're basically doing the same thing as the mipmapping, right?

Alex Soklev Yeah.

Chris Nichols So, you have something that's going in that way, but we have different ways of doing things. So, you mentioned ... and people who have used V-Ray for a long time have seen the term static memory versus dynamic memory, which is, I'm sure where we're going to go here. Static geometry or dynamic geometry. So, let's define what that is, and what is the advantage of one over the other? Because I think that's how it all starts, right?

Alex Soklev Yeah. So, from the beginning of V-Ray, there's been two types of geometry. So, there's static geometry and there's dynamic geometry, or at least that's what we call them. So, they're very different while they do serve the same purpose. So, it's all geometry, it's just the acceleration structure that V-Ray intersects in order to find in the fastest way possible, but hit along the distance of the ray so we can share that. There are many ways, but there are two ways that we intersect with geometry.

Alex Soklev One is when you put all your triangles into one and the same acceleration structure. So, all your meshes, you group them together and you build one huge acceleration structure, so all the triangles. This is what we call static geometry.

Chris Nichols One big tree.

Alex Soklev Yeah, one big tree over all your triangles, across all your meshes. This is what we call static geometry. So, all your static geometries in V-Ray will go into one single tree. The opposite to this is to have a separate tree for every mesh that you have, and then you build a secondary tree on top of the first one, which would just bounce those mashes. It's called a BVH tree, a bounding volume hierarchy.

Alex Soklev The difference between the two is that static tree, when you build one single acceleration structure over all your triangles, is generally faster than the other one. It's not a big difference, but it's noticeable, especially on bigger scenes.

Alex Soklev So, why do we have dynamic geometry? Well, usually people like to fiddle with their scenes, like in interactive rendering. You start your render, you build your acceleration structure, everything's working perfectly. And now you want to move a certain piece of geometry. But in the static tree, because everything is in one place, if you move one piece of geometry, you'll have to rebuild the whole tree once over again, even if you just moved it, it's a change in the tree and you have to rebuild the whole thing.

Alex Soklev And because there's probably millions and millions of triangles in that, it's expensive. You need to wait for a few seconds for this whole thing to rebuild. On the other side, you have dynamic geometry where every single geometry is in a tree of its own. So, if you move a tree over, a piece of geometry over, you don't need to rebuild everything, most probably you will have to rebuild only the tree that this geometry represents, which is a smaller thing, like a few hundred thousand or millions of polygons. And you'll have to rebuild the BVH itself, the secondary tree on top of the first one. But it's also very, very small because there's as many nodes in it as there are nodes in your other scene.

Chris Nichols Right.

Alex Soklev So, this is very, very fast to update. And this is why we have both of them, because one of them is good for interactive rendering, the other one is good for final rendering.

Chris Nichols Right.

Alex Soklev And, also the problem of the static tree is that it doesn't pull out instancing. So, if you have a huge piece of geometry, and you want to instance it a few times in the static tree you have to actually copy it a few times. But if you don't want to pay that price, you need to be able to just reuse the geometry. And, in this case, you have to make it dynamic, you have no other choice.

Alex Soklev So, in out-of-core from where we started, you need to be able to rapidly remove things from your scene and bring them back in. And, for that reason, static geometry is not good to have, because every time you need to remove a few triangles from your scene, you have to rebuild the whole tree again and again and again, and this will kill performance right there.

Alex Soklev So, you have to use dynamic geometry even when you go out-of-core. And this is actually something that we are doing. So, if you enable out-of-core, we will no longer put any of the measures in static geometry, everything will be dynamic. And this is another thing that we're doing. We're trying to optimize our dynamic trees as much as possible, so there's no performance gate. Because, as I said, in static geometry, it's a little bit faster.

Chris Nichols But, you're taking the hit for convenience, like, that's okay.

Alex Soklev Yes.

Chris Nichols It's a little bit faster, but you're able to ... you can do evictable memory and all that other stuff, right?

Alex Soklev Yeah.

Chris Nichols Okay.

Alex Soklev And also there's a few tricks up our sleeve, some things that we're doing currently in V-Ray that we'll do with out-of-core as well.

Chris Nichols Okay. But let's bring back the idea. So, now that we've looked at the geometry way of doing things, it's basically like a bunch of mipmaps. So, you have a master tree, which is your BVH tree, right?

Alex Soklev Yeah.

The Holy Grail of GPU rendering

Chris Nichols And then you just load what you need on-demand into your scene and then, et cetera, you're good to go. Very similar to the mipmap level of textures. However, we are also looking at evicting things because some people are saying that they do "out-of-core," but really what they're doing is just on-demand. They're not actually doing evictable memory. That's like what we are defining as out-of-core is when we actually have evictable geometry in that situation, right?

Alex Soklev Yup. So, the difference between the two is, for example, with the skyscraper scene and the city scene that I said. So, if you have on-demand geometry loading, which is just a part of what out-of-core is. So, out-of-core has on-demand geometry loading, but it has much more.

Alex Soklev So, if you have on-demand geometry and you're, for example, doing interactive session in this specific scene with the skyscraper and stuff, you can start rendering, and everything that's in the scene, and everything that's outside through the window will get loaded, and it'll fit in your GPU, for example. But, if you try to move your camera around, and you want to go outside of this room during the session, you'll say, "Let me start hitting more stuff," and you'll need to put more stuff in the memory, more trees, more meshes, everything, and suddenly you're out of memory.

Alex Soklev And, if you just had on-demand loading at this point, you'll crash because there's no way you can remove stuff from the GPU in order to make room for the new stuff, and with out-of-core you can. And another thing, by the way, that out-of-core is very good at is this specific scene where you're in one room, for example, in your set and you move on to another room.

Alex Soklev Now, you can't fit both rooms in the memory, but you're never in both rooms at the same time. So, when you go in the other room for a few cycles, you will be out-of-core because you need to remove all the stuff from the first room and load everything in the second room. But, once you are in the second room and you evict everything from the first room, you're no longer out-of-core again. Everything fits in memory and you can be fast. And this is the Holy Grail, this is what we're after, this is what we want.

Chris Nichols Yeah. It's an interesting thing, but you know, what's also amazing is that the GPUs have gotten faster, obviously, as we know, and they've gotten new special cores like the new RT cores, et cetera. But they've also gotten a huge amount of memory recently. The fact that your standard gaming card can have 11 gigs or 12 gigs of memory is pretty darn good.

Alex Soklev This is pretty darn good, but then you can look at the professional tier cards when you have stuff like the RTX 8000, which gets 48 gigabytes of memory. And you can actually NVLink those together-

Chris Nichols Get 96 gigs.

Alex Soklev Things are getting really, really out of hand. And can you imagine going out-of-core with 96 gigs?

Chris Nichols Well, I'll tell you what? I've tried, and I had a massive scene, and I was like, "Okay, I'm really going to test this NVLink to see how far ..." and it was this massive, massive scene. And, I was sure because on the CPU, it took like over 140 gigs or something like that. It was a ridiculous amount.

Chris Nichols And so, I was like, "Well, this is great, because now I have 96 gigs of memory on the GPU. So, I'm going to be able to do this. I'm going to be able to see how it's going to go out-of-core. The problem was that because I implemented the on-demand textures, you know, the mapping textures. Turns out I only needed 39 gigs of memory on the GPU to render. So, I couldn't even go out-of-core on one of them.

Chris Nichols So, the way we deal with memory, I mean, obviously memory is so important on a GPU, the way we've dealt with it for a long time, is to be really, really smart about how we put it into memory. So, we're already smart in some ways, and the on-demand is going to be a big thing. But even so, even if you start to run out of memory, we still have the ability, with evictable stuff, to continue to render no matter what, right?

Alex Soklev Yeah.

Chris Nichols It won't crash.

Alex Soklev Yeah. This is the goal. We want to make the renderer so resilient, that no matter what you toss at it, it will produce a result, it won't crash. It would render it out, maybe it will take longer if you really, really stress it, like if you have a few gigabytes and you want to render hundreds, yes, it could be slow. But, the goal here is to make it resilient and handle anything.

Chris Nichols Right. Okay. So, I have one more thing. Alex, are you part of the Lavina team at all? Are you working on Lavina?

Alex Soklev No, not at all. I'm familiar with the people working on Lavina, but I'm not on the team.

Chris Nichols I'm just going to ask you to chime in, knowing that you're not actually on the team, but Lavina for us is our real-time ray tracer. What I want to talk about here is, there is something very specific that we are doing that other people who are claiming to have real time ray traces are not doing. And that is that we are doing 100% ray tracing, no rasterized rendering at all.

Alex Soklev Yep, that's true.

Chris Nichols So, one, while this sounds, like, okay, so what's the difference? As long as it looks good, it's good. But there's one thing specifically that we can do that I want to figure out if you can find a good way to explain it. We can load a huge amount of geometry compared to rasterized rendering, and render that in real time. And it almost doesn't even matter. You can just keep throwing more and more and more geometry, and it doesn't slow it down. How is that possible, and why is that only possible with fully ray traced scenes compared to rasterized scenes?

Alex Soklev So, that's actually the definition of ray tracing.

Chris Nichols Oh, okay. There we go. Go back to your school work, right?

Alex Soklev Yeah. Ray tracing and rasterization are two very different techniques for drawing on the screen. So, rasterization was invented with the goal to render a lot of pixels on the screen. So, it's pixel-oriented, let's say. So, it's very fast at drawing pixels, and there's a fixed pipeline, there is hardware along the way. People have done a lot to optimize rasterization over the years. And there's a lot of, as I said, hardware involved in it that's why it's blazingly fast.

Alex Soklev But essentially, in rasterization, when you want to draw a pixel, you have to linearly go through all the triangles in order to find which triangle and you're seeing maps to that big pixel, so you need to render it. So, for every pixel you need to go linearly through every triangle out there.

Chris Nichols Every single one?

Alex Soklev Yeah. And when there are many triangles in the scene, rasterization starts to suffer. And this is why in games, for example, you're limited by polycount, and everything that's using rasterization is limited by polycount because your operations are linear. For every operation, you need to go through every single triangle to check. While in ray tracing, you have acceleration structures, you have trees. So, when you have a ray and you want to find which triangle it hits, you just can divide your triangles into two big boxes, like put them into two boxes and check whether your ray intersects any of the boxes.

Alex Soklev So, if it intersects the first one but doesn't intersect the second one, you can directly eliminate half of your scene. And this process repeats the second time you remove half of what's left of your scene, and then you remove the half of what's left of your scene. If you have 1,000 triangles in just 10 steps, you can find the one that you're after. And in rasterization, you need to do 1,000 steps.

Alex Soklev Now, rasterization is fast and because it does that in hardware, and up to this point ray tracing had to do it in software. But now with Lavina and with RTX, you can do that in hardware as well. So, you have all those triangles, but you can intersect them blazingly fast.

Chris Nichols Interesting.

Alex Soklev In Optix, something that Optix also does, it can compress the geometry, it can reduce the memory footprint on the geometry and expand it in hardware later. So, it can actually take less memory than rasterization and be faster than rasterization. This is why you can put so many triangles there and have real time performance.

Chris Nichols That's amazing. Okay. Now, we should also note that these RT cores are the first, these are the beginning of them. These are the first generation of this, right?

Alex Soklev Yeah.

Chris Nichols So, we're at the beginning of ray tracing, and that is going to get faster and faster from now on, right?

Alex Soklev Yup. It'll definitely be faster. And, the interesting thing here is that ray tracing is comprised of two parts. There's tracing and there is shading. So, if you just remove the tracing, you're still left with the shading. So, if your scene, for example, takes one hour to render, 50% of the time is tracing, and 50% of the time is shading.

Alex Soklev If you make intersections so fast that it's free, you're still left with a 30 minutes render, because you still have to pay the price for the shading. But, I am very optimistic that now that ray tracing is so fast and intersection ray tracing is so fast, there's a lot of good things for us in the future for shading. So, the next thing to do is make shading fast.

Chris Nichols Okay, that's good. That's good to know. Now, let's talk about one more thing. On these things, especially in video games, you're seeing real time ray tracing, right?

Alex Soklev Yeah.

Chris Nichols And they're really, basically all they're adding is just reflections on top of rasterization. But what they're really doing is you have to render it twice now. You have to render it as a rasterized version and then put ray tracing on top, right?

Alex Soklev This is true, but for game engines it's not a new thing, because they still had to render twice the reflections. And even though they had to render twice, so they do one pass without the reflections, and then they do a second pass where they just position the camera at the location of the mirror, for example, so they can see what's on the other side so they can blend them both together. This is how reflections were done before in games.

Alex Soklev For every bounce you had to reposition your camera or render a new scene, and then you had to reposition the camera or render a new scene. So, this is very, very expensive. You can't do it all the time, and you bet ... for every mirror, actually you'd have to report every mirror, and never reflect the surface.

Alex Soklev So, you have to be very careful what's reflective in your games and what's not. And now you can do it just like that because it's ray tracing. This is what Ray tracing is all about.

Chris Nichols But if you did fully ray traced, and you just don't even have to worry about anything, then everything is just one geometry, one thing, et cetera, right?

Alex Soklev Yeah. And with games they also need to maintain a steady FPS with all that going on. So, for example, you're not doing full ray tracing, you're doing ray tracing but you're also doing rasterizations for your game. And you're engaging the hardware in two different aspects here. It's already stressed out enough with rasterization, so you don't have a lot of room for ray tracing out there.

Alex Soklev So, they're very, very careful on how much they ray trace, what they do with ray tracing. Actually, instead of doing full ray tracing, they do just a few samples in order to sample, for example, the reflection and have a general idea of how it should look. And then they apply denoising and all that stuff to recreate the rest of the image that they don't have in order to provide the visual feedback in real time. While with ray tracing, you just fully go all in with ray tracing on a GPU and you can do everything.

Chris Nichols Yes. And that's a big difference, I think. People should realize what those differences are, because when we're looking at real time ray tracing, real time ray tracing for us is fully ray traced. We are a ray tracer, we don't ride a rasterized engine. So, we are going fully ray traced in Lavina, and that's a big difference. And that's why we can do huge amounts of geometry, much more than any kind of rasterize engine can do, as well as put in global illumination, real reflections, real glossy reflections and everything else, and it looks really good.

Chris Nichols So, this is cool. Well, listen, Alex, this has been really interesting. And I'm very excited about to know where out-of-core is going, and the fact that it's coming. And you're able to talk about it at GTC, we'll make sure to put the link in the show notes for that. And I'm always happy to check in with the GPU team and see where that's going because that's always a big story for us.

Chris Nichols But, I really appreciate it. But, we will definitely have you back on to continue talking about this, because this is always something that I'm fascinated with. And when we talk about pure Ray tracing, it's like, "Huh." Yeah, you really start to think about what the implementations are, and how cool ... It is really cool technology. And it was fun to hear your story about like, you weren't really interested in programming until you finally figured out, "Ray tracing, this is cool."

Alex Soklev Yeah, definitely. It's the coolest thing out there.