Tony [00:00:04] Welcome to Code Together, a podcast for developers by developers, where we discuss technology and trends in industry.

Tony [00:00:11] I'm your host Tony Mongkolsmai.

Tony [00:00:18] Over the last couple of decades, the world of computer graphics has been growing at an amazing rate. From animated movies to video games, we consumers are lucky enough to reap the benefits of these technological advances. Of course, as the experiences get better, the expectations also go up.

Tony [00:00:32] One of the latest algorithmic advances in rendering is path guiding. Today we're going to learn about path guiding and how you can take advantage of this cool technology. To that end, we are joined by two developers who have been working on bringing path guiding to the Chaos V-Ray renderer through integration of the Intel Open Path Guiding Library or OpenPGL.

Tony [00:00:49] Dian Nikolov has worked on multiple render engines at chaos, including V-Ray. He has worked on features like progressive caustics, toon material and various V-Ray GPU and Vantage features. Of course, one of his latest works is integrating Intel OpenPGL into V-Ray. Welcome to the podcast Dian.

Dian [00:01:11] Thank you.

Tony [00:01:11] We're also joined by Sebastian Herholz, who is a ray tracing engineer at Intel who works on the rendering kernels library team. He's been interested in computer graphics for two decades and is the main developer and project lead for the Intel Open Path Guiding Library. He has worked to integrate path guiding into large, well-known projects, including Blender and now, of course, Chaos V-Ray. Welcome to the... Welcome to the podcast, Sebastian.

Sebastian [00:01:35] Welcome Tony. Thanks for being here.

Tony [00:01:39] All right. So I am very much not an expert in rendering. It's probably true about most of the things that we have on this podcast. I am not an expert, but fortunately I have both of you guys today. So many people have heard about things like ray tracing. Probably fewer have heard about path tracing. So, Sebastian, can you talk to us and describe to us what path tracing is?

Sebastian [00:02:03] Yeah, sure. So path tracing is a rendering algorithm for physically based rendering, so which tries to simulate like a real physically based like transport in a scene. So when you want to render a scene like a synthetic scene description of a game or in movies and you want to make it look like physically correct or plausible, like this is a rendering algorithm you can use. And what it does is it shoots a path like random paths in the scene, like from the camera. And each of these paths starts at a pixel. And these paths then explore as a scene to explore the light transport and the light distribution of the scene, and to try to collect all the light which actually contributes to each pixel. So each of these paths actually estimates the final pixel color. And because it's this estimation can be good or bad, you need a lot of these estimates to average them to get the final pixel value. And these arrows you have in these estimates, is something we see as noise, in these rendered images. And the more samples you pick and take, the better. This noise goes away and your estimate that you're rendering converges to the real noise free image.

Sebastian [00:03:26] The big problem here is how does the path tracer and make the decision to explore the scenes to collect all the light. And what is this path tracer can do is it hits like the surface and then makes a random decision where to go next to explore the scene. One shortcoming of these path tracers is that usually you do not have like any information about the complete indirect light distribution of the scene because this is actually what the path tracer tries to solve. It tries to calculate this light distribution, but you don't have it. So the only thing a path tracer usually does is, when making a new decision for a direction, is looking at the local information it has, which is the material at the current intersection point. And then it looks at the scattering distribution of this material and shoots a ray and continues the path in a direction where, if light would come from this direction, the material would scatter a lot of light towards the camera. And this is how a path tracer constructs these paths to exploring the scenes. This works quite well when you have not huge variations in the indirect light distribution. So there you actually get can render nice images with few amounts of samples. But these algorithms really struggle as soon as you have complex indirect light situations such as caustics or really multiple bounce, diffuse light. And this is where algorithms like path guiding can help you. But probably this is something we continue talking later a little bit in more detail.

Tony [00:05:08] For the person who doesn't think about this all the time, I'll come with the direction of somebody who looks at this kind of mostly in games or in movies. When I look at what is being rendered, what you're saying is that path tracing/path guiding helps us with understanding how much light is in each part of the scene. Because really, when I look around the room, for instance, that I'm in right now, certain parts of the scene are brighter or darker and through path tracing it helps me understand as light reflects off various different surfaces. The illumination will change in different parts of the room, and what part tracing does is help me get a more accurate idea of how much light is in each part of my room in some sense, right, By looking at how it bounces off different surfaces because that behaves differently?

Sebastian [00:05:56] Yeah. This is how it works like, so this is what light it does. It starts usually from your light source and then it bounces multiple times on all the surfaces until it at some point reaches your eye. And what path tracing tries to do is to reverse actually the path of light. And so what you usually try, for example, in games is to use an approximation of the indirect light. So you have like some probes which you rendered before, and then you try to use these probes to say, okay, this is the amount of indirect illumination I have here, and it looks plausible and good enough, but it's often not a real physically, correct result you get.

Tony [00:06:39] That's what brings us kind of to Dian and Chaos V-Ray. Dion, if you can explain what that is and why that's different than what I get when I'm playing a video game.

Dian [00:06:47] Yeah, sure. First of all, thanks for having me. I like how this explanation started really simple. Like, yeah, we should shoot some rays from the camera in the scene for each pixel. And then we estimate the contribution. But then, then the problem starts coming in like you don't know where the light comes from. You only have local information and so on. And in theory, when you first first see the path tracing algorithm, it's quite simple. But eventually you realize that it's simple only if you are able to trace infinite amount of rays. At the moment, we are not even close to that. V-Ray, similar to path tracers in the world, has this problem that it cannot trace an infinite amount of rays. So it has to be smart when choosing in which directions to trace rays. The latest thing that helps us is Intel's library OpenPGL we can talk about it in detail.

Tony [00:07:52] So, Sebastian, when we are talking about how OpenPGL is helping in this particular case, the Chaos V-Ray renderer what is the advantage of OpenPGL over traditional path tracing?

Sebastian [00:08:08] Yeah, so as I mentioned before, what a path tracer usually does to decide where to go next is just to look for the local material and the reflection properties of this material. But it completely neglects or has no information about the incoming light at this point. So it could mean, yes, you have like a really glossy, shiny surface which gives you like more of a cone shape of reflection in one direction. But there's a super bright light spot or light source or indirect light source not coming from this cone, meaning this is the direction you maybe should go, but because you don't have this information, but there is actually some indirect light coming from another direction and this primary cone of of reflection and you have a low probability to actually exploring this direction, which increases the noise of your renderer.

Sebastian [00:09:05] So the idea of path guiding is to say during rendering, we try to learn an approximation of the indirect light field and then we can use this approximation to actually guide the path tracer into directions which are important or have high contribution to your final image and can significantly reduce the noise of your renderer. And this helps you that you need less samples to get to a noise free image and also helps you to save some time in your rendering. Examples are like multiple bounce, diffused indirect illuminations. Sometimes also some simpler caustics, like if you think about a pool with a water surface that you have some surface on ground and then usually you have like these nice caustics on the ground and these are hard to sample by a standard path tracer because you have probably a diffuse surface which reflects light in all directions, but the energy of the sun is bounded by the water surface to only one small or directional area of the surface. And, shooting array in this specific area is pretty hard. And by learning these indirect directions in OpenPGL, we are able to tell the renderer, "hey explore this direction a little bit more often. Something interesting is coming from there.

Dian [00:10:40] Be smarter with the race, with the directions. Right?

Sebastian [00:10:44] Yeah.

Tony [00:10:45] And you mentioned the word caustics a couple of times, and I think it'd be good to help people kind of understand if we're talking about tracing rays of light. So if I think about simple things like a wall, a flat wall, I can easily say, hey, this, this wall is doing something. But caustic is actually kind of a reflective surface or refractive surface, right? It's as the light travels through something that is not necessarily straightforward.

Dian [00:11:11] Usually the most common case is where when the light bounces from a highly reflective or refractive surface and then focuses on a on a diffuse surface. Then you see like a bright, very bright spot or a very interesting bright pattern on the diffuse surface.

Tony [00:11:37] Okay. So like, for example, in the real world, if I if it reflected like maybe through a piece of glass and then hit a table or something like that, it might come up with something interesting. Or like in my head, I think like maybe like a prism as like, goes through a prism, I get kind of this diffuse image.

Dian [00:11:54] Yes, exactly.

Sebastian [00:11:55] Yeah. Or like a lamp or a glass where through the curvature of the refractive surface or reflective surface, like it actually bundled and focused to one specific point. And this makes this one point or direction way more important.

Tony [00:12:15] We talked a little bit about Chaos V-Ray, but, you know, when people I'll say again, me, when me...when I think about things in terms of rendering, I tend to think of things like video games, like the Unreal Engine. In the old days, the Quake engine, kind of these real time video game type renderings. And then I do think about movies a little bit, like the animated movies that I see in the world. What is the difference between a product like the Chaos V-Ray Render and something that I would see that is rendering, say, a video game? What's the difference between a production level renderer versus like a video game rendering?

Dian [00:12:53] We can talk about real time rendering and and and general offline rendering the differences between the two. And then we can talk about specific things in V-Ray. First of all, I want to mention that real time rendering is very quickly catching up with the capabilities compared to offline rendering in many of the problematic situations. And the good thing about real time rendering is that you can instantly see the result of your scene. You can instantly make changes, you can play around with it and stuff like this. With offline rendering, you have to wait for all of this from maybe several, several seconds to potentially minutes or hours if your scene is very heavy. But yeah, all of these have tradeoffs. But for example, real time rendering has made very significant improvements in the recent years. For example, it can now handle scenes with very heavy geometry, with tons of triangles. For example, in Unreal Engine 5 with with Nanite. Another big problem is how to handle hundreds or thousands of lights in the scene efficiently. And this is also handled now in real time with the resampled, important sampling combined with the reservoir sampling. This is called for example RTX-DI. Real time rendering also utilizes denoising algorithms and Intel also has a denoiser, that can work with very noisy images to produce a very clean result similar to what what you would expect from a few minutes or hours of brute force path tracing by an offline render.

Dian [00:14:52] Another thing that real time rendering struggles a lot with usually is large resolutions. For example, 4K or 8K render resolutions. But now we have upscaling, Intel XeSS, and these are very significant improvements. But there are still areas where real time rendering struggles a lot. And one of the things that we already talked about is caustics. Even if you use ray tracing or path tracing, you won't get very clean results. You have to be a lot smarter than that. And you can either figure out some kind of hack that will work on specific types of scenes, or you can do what an offline rendering will do and just trace a lot of rays and try to find the caustic contribution correctly. Other problems for real time rendering, realistic rendering of skin, of fog. For example, fog smoke as well. Hair shading is problematic. You can have very complex materials that have very complex calculations. You can have procedural textures, stuff like this. These things are very heavy to compute. For all of these things, real time rendering has some kind of workarounds for it, but usually they break down in the scenarios or the scenes they are not intended to work. And it happens quite a lot. On the other hand, offline rendering can can handle on all of these effects.

Tony [00:16:40] So then it sounds like that the difference there is that with the real time renderer, there are corner cases that are just kind of ignored because the person who's building the engine recognizes that those corner cases may not be applicable for the usage of the render, or they leave it up to the person who's using that render to make sure that those corner cases are handled in kind of a one off way. Whereas something like a

Chaos V-Ray will actually render things without needing to cut these corners because it's less worried about making sure the frame hits the screen at the right time, but more about making sure the the scene is accurately rendered.

Dian [00:17:21] Yeah, exactly. It's it's up to the user to to try and find a way to work around or to prepare their scene so it doesn't it doesn't break down with these features.

Tony [00:17:43] And so then going back to the OpenPGL. Sebastian, can you talk a little bit about what makes up the path guiding library? So my understanding is it's not something that Intel's come up with from scratch. Where we've looked at and said, "We understand and let's go find the solutions" But as my understanding is that it's kind of a, what's the right word, amalgamation of best known methods. So can you talk a little bit about that and how we chose these to integrate into OpenPGL?

Sebastian [00:18:09] You're right, Tony. Like path guiding is nothing we came up with. It is actually a rendering technique which is quite old, like the first research papers about it, what was in the early 90s by Jensen and Lafortune like they both had an idea of, "Hey, there are paths which are hard to sample. Let's try to learn something." And we are here talking about an age where people were happy to be able to render a Cornell box like which is a super simple box in a 64x64 image resolution and the machines had about like 32 megabytes of RAM. So they tried to figure out an algorithm which can store some information about the light transport in these super simple scenes and then reuse this. And yes, it worked and they were super happy. But it was also like, yes, it was just a huge amount of memory consumption and compute speed. And so like the whole research area in rendering went more into let's, let's try to figure out algorithmically other ways to make better sampling decisions, like bi-direction path tracing, metropolis light transport, photon mapping like all these algorithms which try to just come up with other ways to make better sampling decisions. And around 2014, years have passed. We have better compute, more compute and more memory. There was a paper by Jirka Vorba who more,more or less, resurrected the idea of path guiding. There have been some smaller works before, but this project is a work from Jirka Vorba was the one which really resurrected it and also showed it in more production context. And then just the whole idea of path guiding...I can learn something either through a pre-processing step or doing my rendering about the light transport, store it and reuse it came up again. And till then more and more path guiding algorithms came up.

Sebastian [00:20:27] And this actually brings us to the problem. If you are a developer and you want to integrate path guiding, there is a huge amount of different papers, a lot from the scientific realm of rendering. So people try it out on small subset of scenes. It works really nice in these scenes, but the big question is, okay, if I want to have a production renderer, which of these huge variety of different algorithms should I pick? And this is where Intel came up with the idea of, "Hey, let's make a library out of it." Like, let's try to to look at all these papers and pick the ones which are interesting and which are promising and try to extend them in a way so that they can be robust enough to be used in production rendering. Yeah, we have two different papers which are the main contributions we implemented. One is a paper I did together with a student, Lukas Ruppert, together we presented at SIGGRAPH two years ago, which is called the parallax aware mixture modes for path guiding. And another one is by Thomas Müller, which is the practical path guiding paper. And we took these two algorithms and and integrated them into the library and tested them on a lot of production scenes. And because OpenPGL is now open source, we also hope that by adoptions of companies like Chaos and V-Ray, we also get a lot of feedback from the real production context and then also can really make it bulletproof so

Commented [1]: Lukas Rupper:
Paper: Robust Fitting of Parallax-Aware Mixtures for Path Guiding
Link :https://dl.acm.org/doi/10.1145/3386569.3392421

that we see all the problems people have in production and can update the library to make it way more robust. So the basic idea is that what Embree is for ray tracing in the professional rendering realm, OpenPGL, hopefully will become for path guiding.

Tony [00:22:27] And then, little plug, Embree is our ray tracing library, which has won an Academy Award, which we'd like to talk about. So that's kind of a cool thing.

Dian [00:22:37] We are also using Embree.

Tony [00:22:39] Oh, there you go. So getting back to how V-Ray supports OpenPGL. How is the OpenPGL library used within V-Ray? Is there a certain thing... Is there a switch I need to turn on to make it work? What types of things should I expect to be supported within V-Ray if I want to use the OpenPGL library?

Dian [00:23:01] Yeah. First of all, you have to you have to understand that the V-Ray is a very, has been around for quite a while, for more than 20 years and is used by all kinds of 3-D artists and designers all over the world. So the scenes that we that we have, and that are usually rendered with V-Ray, can vary quite a bit. And over the years as well, V-Ray has accumulated an absurd amount of features. For the initial implementation, we chose to make it very simple, just a toggle. And we thought that's a good place for that would be to connect it somehow to our already existing light cache prepass because, as Sebastian mentioned, for path guiding, you have to do some training, you have to provide some training samples to the library and then you'll learn the distribution of the lights in the in the scene.

[00:24:12] This is very similar to what our light cache prepass does. It shoots a lot of rays, traces very, very long ray paths in the scene and gathers all kinds of lighting information. And then what we do is just take this information and pass it to OpenPGL as well. And that's how that's how we train it. After that can becomes the harder part to do the actual rendering with and shading with the path guiding. This effects basically the whole raytracing code and the whole shading code.

Tony [00:24:55] So you mentioned that it's a toggle, I guess it's an experimental feature, is my understanding, within the latest release of Chaos V-Ray. Yeah. Okay. So it's it's not something that it's just kind of on by default, most likely. So it's something people need to go in, turn it on and try it. But obviously, you probably don't get everything for free. There are probably things that I need to worry about if I'm trying to render something. So what are the things as a user of this experimental feature that I should be careful of?

Tony [00:25:27] It helps only with certain types of noise. So the goal is basically to to to clean up the noise in the image faster. But it helps only with certain types of noise. And that's that is the kind of noise that comes from basically global illumination, diffuse global illumination and glossy reflections. So that's where it can help and it depends on your scene. You can have noise from various sources in the scene, so you can have noise from lights, you can have noise from materials and material textures. You can have noise from motion blur and stuff like this. But yeah, one of the common or the common source of noise is global illumination and reflection. So, yeah, if you have a lot of that, this can happen for example in interior scenes or scenes with a lot of occlusion. So where we have a lot of shadowed areas where light has difficulty reaching. So you can expect good results there. But for the rest of the scenes you might have, it might just add some overhead. And that's why it's still disabled by default. Ideally, yes, ideally, we would love to to enable it by default and have it always on, but for the first version we decided to do it like this.

Tony [00:27:00] Yeah. In our content creation applications from things like 3ds Max, Maya, Houdini. There are a lot of different places where V-Ray could be exposed. Where is it exposed right now? Is it if I pick up any of these things will work or is there a certain place I need to go to go try out the latest V-Ray with OpenPGL support?

Dian [00:27:20] So we just released it in the new update of V-Ray for 3ds Max. And this is as usual the first update, the first release of the year. Soon, of course, it will be coming for the other integrations like Maya, Cinema, Houdini, and so on. Expect it everywhere in a few months.

Tony [00:27:43] That's awesome. And then, Sebastian, what types of new features or functionality should we expect from OpenPGL? Are we done with OpenPGL and we're happy, or are there new features that we expect to come down the pipeline? You mentioned that we basically took algorithms from two papers and put them into OpenPGL. I guess what's next for you and OpenPGL?

Sebastian [00:28:04] Yeah. So currently OpenPGL is still in a beta phase. So we just started with our journey. So probably end of the month we plan to have we leave beta phase and go to an 0.5 release and this is like the basic start of doing things. We offer users to have a guiding library with guiding structures to guide like direction, sampling decisions in volume and on surfaces. But we also have some plans to to extend it even further. Currently we can only guide directional decisions, but there are also features like something called Russian roulette, which is stochastic path termination. So because usually you would need to have paths with an infinite amount, with an infinite length to be unbiased because light can bounce an infinite amount of time before it reaches your eye. So in theory you would also try to trace infinite lengths of path. But what you do in practice is you have some stochastic approach which more or less gambles at every bounce if the path should be terminated or not. This is usually based on some heuristics which say, okay, if I bounce multiple times from a dark surface, I don't expect that a lot of light comes comes from from these type of paths. So I terminate this path. Unfortunately, this heuristic has no idea if yes, you bounce multiple times on a dark surface, but in the end you hit the sun or a laser beam. And so this can still be an important path, even if it bounces on multiple dark surfaces. And there is a technique called guided Russian roulette or adjoint-driven Russian roulette, which is one thing we want to also give support for, because this can really help you to keep important paths alive and to terminate unimportant paths early and to also increases the efficiency a lot. And this is, for example, one way how you can make up for the overhead you get through OpenPGL with learning all of these distribution and representations in scenes where probably light transport is not that complicated, but you probably still want, can gain speed back from making good Russian roulette decisions. So guided Russian roulette is one of the next things we are currently working on and we hope to to put it out in one of the next releases. And also now said V-Ray has support for OpenPGL, we also hope to get a lot of feedback from the users of V-Ray to see where are we good and what can we do better in OpenPGL also together with developers of V-Ray is to figure out how can we work on the integration into renderers like V-Ray and how can we optimize this so ideally is that it's an always on solution.

Tony [00:31:09] And one question I'm sure people are wondering about is GPU support. So traditionally offline rendering has mostly been done on CPUs and CPU farms, which may be counterintuitive to some, but we're seeing that change a bit. Are there particular aspects of OpenPGL that lend itself to running on a GPU?

Sebastian [00:31:31] So the plan is to make a GPU port and to support GPU rendering, especially since more and more renderers are actually moving or at least to support also some sort of GPU rendering. Sometimes like if you have smaller scenes or you just want to make look depth and you then it's nice to have the GPU which is faster and rendering and then you take the whole scene with all the complexity and throw it on on a big CPU farm to do the final calculations. So this is also why we want to support GPU rendering. And yeah, like the current idea is why we stick to the CPU both that it's easier to debug and work on all these algorithms. And to figure out first okay, if people from the professional rendering area on which are mainly CPU based are happy with the library, then we found a way and an algorithm which is worth porting to the GPU. Instead of trying to do both at the same time, you usually have some problems that programming on the GPU is pretty much different than programing on a CPU. Debugging is way harder and you would start trying to cuff yourself to this other other ecosystem and you might not try to explore everything you want.

Dian [00:32:59] So you might lose a of time dealing with the GPU specific problems.

Tony [00:33:07] Yeah, that's one of the nice things that oneAPI tries to target by allowing us to program once and debug it on a CPU and then eventually target a GPU to actually do the work. And on that note, I think that probably ends our time for today. I'd like to thank Dian for joining us.

Dian [00:33:26] Thanks for having me.

Tony [00:33:28] And thanks to Sebastian as well.

Sebastian [00:33:30] Yeah. Thank you, Tony, for inviting us. It's nice talking to you.

Tony [00:33:34] And thank you, our listener, for joining us. And if you're interested, we actually have a little segment after the music ends where we talk a little bit about how development and work actually gets done at Intel and Chaos sometimes.

Dian [00:33:53] By the way, thanks to Sebastian, special thanks, for helping me with the integration. For sure we wouldn't be able to ship it this soon or for this release if it wasn't for his help.

Sebastian [00:34:09] It actually was a really interesting thing how we did it. I don't know, Dion. You would probably want to tell us a bit about the renderthon. Like what you have every year at Chaos?

Dian [00:34:20] Sure, sure, sure. So at Chaos each year we have this kind of event called renderthon where people from all of our offices around the world can participate and we can develop together various interesting topics and work together on some interesting ideas. And this time, we had a last year we had colleagues from Bulgaria, from Czech Republic and from Germany, from all of those offices gathered in Prague. And we worked on maybe 12 or 15 topics, something like this. We split the into teams of a couple of people and yeah, just work, work all day for three days straight. And see what we come up with. And me and Sebastian and a couple of other guys worked on the initial prototype of integrating OpenPGL in V-Ray. I had to do some preparation because it's very easy to mess up, to mess up the calculations there. And if you don't get them right, you cannot present anything. You you cannot do meaningful comparisons. Yeah. It was a lot of pressure actually to to implement the prototype, but it all worked out in the end and the

results looked very promising. So that's why why we wanted to to try and bring the the feature to the users as soon as possible.

Sebastian [00:36:31] It was an interesting event. It was really intensive three days of sitting together with Dian and some of his colleagues and getting everything done. But interestingly, the POC was done after three days and one or one half months later, suddenly Dian came to us and said, Hey, we will have an experimental feature actually in the next release. And this was pretty good work to continue doing that. And so we are looking forward about the results and how the community actually will receive it and react to it.

Tony [00:37:10] Awesome. And I think our developer listeners will really appreciate that. At Intel, we used to call those, I guess in some cases we would call them dungeons where we just locked everybody in the room until some work got done.