EPISODE 864

[INTRODUCTION]

[0:00:00.3] JeM: Google Earth allows users to explore the imagery of the real-world. Imagery for Google Earth is taken from satellites, cars equipped with cameras and other sources. Google Earth renders a data intensive 3D model of the world on a client application, such as a desktop browser, or a virtual reality system.

WebAssembly is a runtime for executing code other than JavaScript in a browser-based environment. WebAssembly is useful for data-intensive workloads and developers can use programming languages, such as Rust, or C++ in the browser by compiling to WebAssembly.

Jordon Mears works on Google Earth and he joins the show to talk about the engineering behind Google Earth and how WebAssembly is being used to improve efficiency. Jordon also discusses the state of tooling around WebAssembly today.

To find all of our episodes about WebAssembly, you could check out the Software Daily app for iOS. It contains all of our old episodes, all 1,000 of them and you can sort by topics and greatest-hits and you can find related links to each of our episodes, if you're looking for some complimentary reading material. You can also comment on episodes, you can start discussions with people and you can become a paid subscriber if you're looking for ad-free episodes. You can get those ad-free episodes by going to softwareengineeringdaily.com/subscribe. We will have the Android app soon, but in the meantime if you're an iOS developer, please check out the Software Daily app if you're interested.

[SPONSOR MESSAGE]

[0:01:47.8] JeM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens and we don't like doing whiteboard problems and working on tedious take-home projects. Everyone knows the software hiring process is not perfect, but what's the alternative? Triplebyte is the alternative. Triplebyte is a platform for finding a great software job faster.

Triplebyte works with 400-plus tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz. After the quiz, you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple on-site interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte, because you used the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more, since those multiple on-site interviews would put you in a great position to potentially get multiple offers. Then you could figure out what your salary actually should be.

Triplebyte does not look at candidates' backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. I'm a huge fan of that aspect of their model. This means that they work with lots of people from non-traditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple on-site interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out. Thank you to Triplebyte.

[INTERVIEW]

[0:04:07.4] JeM: Jordon Mears, welcome to Software Engineering Daily.

[0:04:09.4] JoM: Thank you.

[0:04:10.6] JeM: I'd like us to spend this show exploring the engineering of Google Earth. I think it will make sense to give some overview of the backend, talk about the middleware. Then once that sets us up for a conversation about the frontend, we'll be able to talk about WebAssembly in more detail. Because I feel Google Earth is an application where the backend and the middleware are just as important to understanding the implications of WebAssembly as the

frontend. Before we actually get into the engineering, let's talk a little bit about Google Earth at a high-level. What is Google Earth?

[0:04:51.6] JoM: That's a good question, because Google Earth is a very interesting product, especially at Google, is that it's basically an application that allows you to explore almost anywhere on the planet and view imagery, satellite 3D imagery and so forth and visit places that you would never be able to travel. I like to use the analogy of it's a video game of the real world. Especially at Google, it's a product that's very different from some of our other products in the sense that it's really just Google's gift to the world, and working to support the mission of organizing the world's information and making it universally available and useful. I probably botched that mission statement, but you're going to have just – Yeah.

[0:05:37.5] JeM: How do people use it? How do people use Google Earth?

[0:05:40.1] JoM: Google Earth has actually been around for about 14 years at this point. As a result, it has quite a bit of legacy. You can actually access Google Earth in four different ways. The original application itself is actually a download and installable client on Mac, Windows and Linux. You can actually still get that client and use that client today.

However, about four years ago or so, we embarked on a project to just reimagine Google Earth at large. In that about two years ago, we actually launched Google Earth on the web for the first time ever, as well as revamped our Android and iOS clients. You can access it on very, quite a few variety of platforms. However, depending on what platform you get, the experience is a little bit different, but the overall gist of experiences there on every platform.

[0:06:29.7] JeM: How does Google Earth differ from Google Maps?

[0:06:33.0] JoM: Yeah, we get this question a lot and I can't tell you how many times I've met people here that I work on Google Earth. They're like, "Yeah, I use Google Earth every day," and they whip out their phone and show me Google Maps and it's like, "Great. Yeah." Then I have to explain the difference.

The difference between Google Earth and Google Maps is Google Maps is about finding your way and getting things done. I think Google Earth is about getting lost. The idea is that it's really meant for exploration, learning about the world, seeing something new and that thing. Whereas, maps is more about where's the nearest restaurant, how do I get driving directions from point A to point B and how do I do things geospatially in my normal day-to-day life? We really focus on that.

I mean, Google Earth itself is wildly used for all kinds of things, everything from – it's very heavily used in education and learning the world and at schools, but it's also used a lot in things like crisis response and real estate and solar panel planning and all kinds of crazy things that you'd never think about, that maps just isn't suited for, because you need that focus on the satellite imagery and less the – what I like to call the paper map.

[0:07:42.3] JeM: Give me a brief history of the engineering of Google Earth. How has the software stack evolved over those 14 years?

[0:07:49.9] JoM: Google Earth was actually originally an external product. It was developed by a company called Keyhole that Google acquired and then relaunched. What's funny about it is have you ever played the video game Crash Bandicoot?

[0:08:03.8] JeM: Of course.

[0:08:05.0] JoM: The original engine for Keyhole and Google Earth was based on that rendering engine that was used in that video game and video games like that, called Intrinsic Alchemy.

[0:08:15.1] JeM: Oh.

[0:08:15.4] JoM: That client today actually still uses all that same technology. The desktop client, it's all C++ based. It's cross-platform using the Qt framework for the UI and so forth. That's still true today and that was true very early in Google Earth's history, all the way to now. We still develop that client and we use cross-platform, or we use that same rendering engine

and so forth. However, about six years ago we decided to diverge and say, we need to reboot the technology.

We actually stopped and restarted the Google Earth and all of its actual engine. It's still all C++ primarily, but we started a number of new projects. There's a open source project that Google sponsored, is called Ion. I don't know if you've ever heard of it, but that actually does a layer of abstraction over OpenGL, that allows us to compile across platforms. That includes everything from Windows, Linux, Mac, Android, iOS and now obviously the web through Native Client and WebAssembly. About 60 years ago, we built this project called Ion. On top of that, we've rebuilt our renderer from the ground up and founded just really a whole new code base for the client.

[0:09:27.4] JeM: Well, that sounds a pretty useful abstraction, because the core problem of Google Earth is in some sense rendering this 3D model of the world. You can forecast that technology for rendering stuff is going to just improve. A core problem of just business logic of rendering stuff on the page, you're probably going to want to do – I don't know. It seems like a useful abstraction, because you can make updates, these lower-level updates, like you have in WebAssembly here at that Ion interface level.

[0:10:03.1] JoM: Yeah, it's been easier for us to port to different platforms. A great example of that is actually Native Client, WebAssembly and we also actually used to have a JS build that did function, because of that abstraction. Another great example of where this is helping us is that with Apple moving away from OpenGL and on to metal, or whatever it's called, that actually poses a problem for us, because if we want to keep producing our iOS application, we need to port to yet another graphics rendering API. Yeah, we're actually doing that. We're at our lower level and not really impacting the overall client code itself and so forth. We're working through that as well right now.

[0:10:45.1] JeM: That's cool. A side note about Crash Bandicoot, there's a really good Quora answer somewhere about some of the engineering problems with Crash Bandicoot. I highly recommend checking that out if you're interested in vintage PlayStation engineering problems. Random, so I don't – I go to g.co/earth, that's the Google Earth short link. There is some load time when I go there. There's a little progress bar. Stuff is loading. It doesn't surprise me,

because there's a lot of stuff going on in the Google Earth application. What's going on as that progress bar is sliding along? What's loading on my browser?

[0:11:28.3] JoM: Currently, we use Native Client, which is the Chrome-only technology for cross-compiling native code for the web. Specifically, when that loading bar is showing it's actually both downloading that Native Client binary, as well as doing the translation of that into the runtime. Actually if you notice, if you look closer, you'll notice that it freezes. The loader freezes about two-thirds, three-thirds of the way through. That's actually when the code is starting up in your browser and then it just – we have no way to measure what progress is happening there, so we can't really indicate what's going on and then it just disappears when it loads. That's what is doing right now.

You're right, because Google Earth is really a heavyweight application. It's a video game of the real world, and so there's a lot. Our binary size for the web is unusually large and there's a lot of threads and so forth that we have to allocate, memory we have to load up. There's a lot going on to get this thing up and running. Yeah, we try to show the user that something happening and give a little joke about quantifying the world. I don't know if you noticed, but if you reload that page over and over again, it actually rotates through, I want to say 12 or 15 different messages of just random ways to quantify the world, everything from human population to grains of sand, to the amount of liters in the ocean, etc.

[0:12:47.2] JeM: When this thing loads, I can take a tour through earth. I can land myself anywhere on the geospatial map and look around at mountains and look at storefronts. It's just a beautiful experience. You can very easily imagine augmented reality, or virtual reality applications built with Google Earth. The engineering problems are also deeply fascinating. If I think about my Google Earth runtime, I know that the entire planet’s imagery is not loaded on my browser at all times. I know that as I move from one place to another in the virtual earth, it's loading maybe adjacent areas, it's probably doing some cache prefetching.

In order to have this data dynamically fetched to this heavyweight browser application, there needs to be some rich client-server relationship where my browser is eagerly fetching data from the server based on some rules, or heuristics. Can you give me more detail on the client-server relationship?

[0:14:01.8] JoM: Yeah. For things like the imagery data, the 3D data and even the what we call the map data, the lines, the labels and stuff like that, so basically as you move around, we're analyzing basically what the camera can see and where it's flying to. We pretty aggressively both send out and cancel and process requests based on where you're going, where you are. You can even see indication of how well resolved the current scene is. If you look down in the bottom-right of your window, you'll see that there's a little spinner that gives you a percentage sign. That's actually representative of how resolved the given scene as I use the data in from the server. Has it been decompressed, processed and actually drawn into the frame.

Basically in broad strokes, what the server provides us in that case is that we have the whole world at various levels of resolution, everything from satellite imagery to aerial imagery, to stuff collected by driving around on the ground, etc. We build this hugely complex and detailed mosaic of the entire world. Then what we do to that is we actually split it up into what we call a quadtree. Basically, as you're zoomed further out, we can take a slice of the world at much lower resolution and it's also much smaller data-wise, so we can load that very quickly. As you zoom in, we just have more and more levels of detail and tiles that represent that detail, all the way down to the ground level, such that as you move around, you're not necessarily loading more data because you're just getting more detailed data of a smaller area, if you follow me.

[0:15:40.1] JeM: Oh, interesting. There is a data structure that is representing a geospatial segment of the earth. That data structure has a – you can go deeper and deeper into it and see more and more granular detail of that data structure.

[0:16:04.0] JoM: Yeah. You can think of it as a pyramid, right? If you think of the whole planet's surface as a pyramid, and if you're down on the ground there's going to be – I don't even know the math, but millions if not billions of individual tiles across the whole surface. If you're zoomed out to space level, there's maybe only 4, or 16, right? As far as the detail in those tiles at the top, where there's only 4 or 16 tiles, the detail is a lot lower, right? It's because what you can perceive and something on a screen that size isn't that high. We don't have to paint the Empire State Building in 3D detail when you're zoomed out that far, for example. If you're in Manhattan, we need a really highly detailed model of that building, right?

[0:16:46.5] JeM: You call that data structure a quadtree?

[0:16:48.8] JoM: Yeah, it's a quadtree. Basically what that means is that it's a pyramid of data that is subdivided by four as you go down. Four becomes 16, 16 becomes 32, etc., etc., all the way to the ground.

[0:17:02.7] JeM: I guess, that defines the level of zoom almost, right? At each increment of zooming in further, I'm seeing a 4X multiple on the granularity?

[0:17:13.0] JoM: Yeah, you could think of it that way. Yeah, we basically maintain an altitude in the camera and that roughly translates to a numerical zoom level, that then that's the set of tiles that the client will pull from the server.

[0:17:27.1] JeM: Cool. Do you pre-create all these quadtrees and then they're just sitting on the backend, or are these created on the fly based on the angle of the earth that I'm looking at?

[0:17:41.4] JoM: They're all actually pre-created. I don't actually work on the data processing and serving, I work more on the client. My understanding is they're actually just all on disk at the time. We do do some transformations of them on the fly as in, but I think that's more just compression formats and stuff, because for example, phones hold the tiles in a different format than necessarily the web app does, that the desktop app does. Yeah, all the tiles are pre-computed beforehand and just on disk. The other interesting thing about that it's not just the current set of tiles of the current picture of the world, we actually have versions of the world from all of this imagery crafter reaching as far back as I want to say the early 1900s in some areas.

There's old aerial pictures of New York that someone took out of some of the first planes that we've been able to go back and find in archives and then process. Now you can actually access those historical tiles as well. You can't do that in today's web application, but that feature is a feature of the desktop application that we're looking to port into the web application at some point.

[SPONSOR MESSAGE]

[0:18:50.8] JeM: As a software engineer, chances are you've crossed paths with MongoDB at some point, whether you're building an app for millions of users, or just figuring out a side business.

As the most popular non-relational database MongoDB is intuitive and incredibly easy for development teams to use. Now with MongoDB Atlas, you can take advantage of MongoDB's flexible document data model as a fully automated cloud service. MongoDB Atlas handles all of the costly database operations and administration tasks that you'd rather not spend time on, like security and high availability and data recovery and monitoring and elastic scaling.

Try MongoDB Atlas today for free, by going to mongodb.com/se to learn more. Go to mongodb.com/se and you can learn more about MongoDB Atlas, as well as support Software Engineering Daily by checking out the new MongoDB Atlas serverless solution for MongoDB. That's mongodb.com/se. Thank you to MongoDB for being a sponsor.

[INTERVIEW CONTINUED]

[0:20:13.6] JeM: Okay, this is a PSA that if anyone from that backend team is interested in coming on the show, I'd love to do an interview with them. If you bump into somebody, you could let them know, because I'm very intrigued by that problem. We've done some shows in the past around these companies that are doing street view backend processing, this emergent set of application backends that are taking photos and stitching them together and building augmented reality systems, or building virtual reality systems, basically building these virtual models of the real world. This is a really tough data engineering problem.

That's not why you're on the show. You're on the show to talk about WebAssembly, so we should get closer to the frontend. I would like to know a little bit about the middleware. To keep the relationship between the backend and the frontend fast, if we just think of this problem of basically fetching the quadtrees, that seems the core problem here is which quadtrees should we be fetching and how should we be streaming them to the frontend? What's the relationship between the frontend to the backend? Do you use protocol buffers, or how is data being shuttled between them?

[0:21:28.3] JoM: Yeah. Generally speaking, everything's transferred in protocol buffer format. That's usually just the metadata though. I don't think the actual bytes of the imagery or transferred in that format. They're actually done I think in DXT or something like that. Don't quote me on that, because –

[0:21:42.5] JeM: TXT?

[0:21:44.2] JoM: DXT, I think is the compression format that we use there. That one I'm not as sure. I don't actually work on that part of the code myself. Yeah, generally speaking communication between our client software and the backend, especially for things like searches and other content, it's all protocol buffers as the wire format. With the actual imagery and compression data, I don't think it's the same case. It's actually just, yeah, some native compression that we use and then decompress on the client. I think DXT is one of the popular ones.

[0:22:13.8] JeM: Can we boil down the responsibilities of the client in terms of data fetching to just grabbing these quadtrees, or is there anything else we should touch on that front before we get to the rendering side of things?

[0:22:25.8] JoM: Well, I mean, rendering imagery is just one piece of what the client does, right? You're able to search for things, you're able to look up content, you're able to read-void your stories, there's all kinds of other things you can do. As far as talking to the server and whatnot, it really just depends on the activity, but the client has to handle making the right request for that. I mean, we use just simple HTTP requests for all of that. There's no socket involved or anything like that. It's very straightforward. Yeah, I mean, it's very simple, honestly.

[0:22:54.2] JeM: Yeah, okay cool. I just wanted to get a contour for people who are less familiar with Google Earth. As we said, this is a resource intensive application. What are the client requirements to run Google Earth?

[0:23:07.4] JoM: Yeah. Google Earth on the web specifically, it does require Chrome and that's due to our use of Native Client of cross-compiling our core C++ for the web. Beyond that, we don't really have strict requirements as far as RAM you should have, how good of a machine

you have, what graphics cards are required. We really delegate a lot of that to just what Chrome the browser itself is able to handle and now to run. However, if you do have some lower-end hardware and so forth, you are probably going to have a bad time.

It's interesting, we even do keep track on our analytics, we do actually track things like frame rate for users, the amount of dropped frames that we're getting. We do actually track analytics on that and we even do collect some statistics on what graphics cards and so forth are the worst behaving ones for our product. By and large, you don't need a super high-end machine to run Google Earth. You just need something that was made in this decade.

[0:24:02.9] JeM: You've mentioned something called Native Client a couple times. What is that?

[0:24:08.8] JoM: Native Client, well maybe I should just back up a little bit. Google Earth itself is implemented, the client itself is implemented all most extensively all in C++ code. What we do is we compile that code across all of our various platforms, web, Android and iOS in this case. The advantage that that gives us is that it keeps us from having to build this application basically three times. I would say 80% to 90% of all of our application logic and code is actually at C++. There's only about an overall surface of maybe 10% to 20% that's actually in each platform.

That's really helpful for us as far as reuse. The other part of it too is that being a native code, generally speaking it's a lot more performant for the user on the device. I think that's especially true on mobile devices. Native Client is a technology that Chrome came out with some years ago that allowed you to cross-compile native code and run it in a browser.

I would equivalent it to some bytecode engine that would run in your browser that seemed not JavaScript or whatever. At the time that we launched Google Earth for the web, it was really the only technology that allowed us to get the performance that we wanted in the browser. Instead of not being to launch a web product, we went ahead and launched it using Native Client in Chrome only.

However, since that time WebAssembly has become the WC3 sponsored and supported standard and all of the major browsers are trying to move towards WebAssembly. We are trying to move ourselves away from Native Client on the WebAssembly as the new way to compile

native code for the web basically, hoping to get some of the same reuse and performance out of that.

[0:25:50.6] JeM: One way to describe WebAssembly and this is not a comprehensive way to describe it, but WebAssembly lets us write modules in languages other than JavaScript and run them in the browser. If you were able to do that with Native Client, why wasn't there a bunch of hype and excitement and tooling and ecosystem stuff built around Native Client?

[0:26:18.0] JoM: I don't know the history of Native Client really well myself. My overall impression is that Native Client was something that Google and Chrome developed, put out to the world and it just didn't see the adoption from the browser community. Other browser manufacturers didn't really pick it up and let it become that standard. Even though it was open source the whole time and I think that's because that it was one of probably a number of ways at the time.

I mean, there's [inaudible 0:26:44.1] with one of the other ways to do this thing, or it would take native code and actually compile it as JavaScript. It was unclear which way it was going to be probably the way that most browsers would go, but the reality came that neither of those took off. Instead, the browser community introduced this thing called WebAssembly and they all rallied around that. Instead of trying to push forward with two standards, Google and the Chrome team just said, “Okay, hey, let's let go of Native Client and really help out getting WebAssembly off the ground.” That seems to be where things are going and we're participating in porting that way.

[0:27:21.1] JeM: Describe what WebAssembly is in your own words.

[0:27:24.0] JoM: WebAssembly for us is a way to allow us to compile, yeah, other things in JavaScript for the web and in our case, it's all C++. The big benefit that gives us is again, largely the code reuse, as well as the performance. My understanding of how it works is that it takes native C++ compiles it down and distance by code style format that the browser can then interpret and then run in the browser.

[0:27:48.3] JeM: Why is that necessary? Just taking a very naive approach, why can't I just do everything in JavaScript? I mean, my browser runs JavaScript just fine. Why do I need these other languages?

[0:28:02.3] JoM: I think that that would be fine if at least for us on Google Earth, if we weren't cross-platform. For example, we have other rendering engines here at Google that rather than the browser that are implemented in pure JavaScript. However for us, we're actually a relatively small engineering team and we want to make Google Earth available to as many people as possible. I think that's our main choice for that. Also, I think there's just some applications that JavaScript itself just can't perform as well as native code can. That's probably the second benefit that we get from that as well, is I think that it's just a lot slower of an experience for a user.

[0:28:41.7] JeM: Before we delve a little bit deeper into WebAssembly, I want to revisit what you said about the rendering stack. You said there's something called Ion, OpenGL is involved. I'd like to better understand where WebAssembly fits into your rendering workflow.

[0:29:01.3] JoM: Yeah. WebAssembly, Native Client, Androids, NDK, it's basically just a compilation target for us. It's a architecture I would say that we target. All of our code is very abstracted into just native C++. Then there's a very thin amount of API-specific code that's written. In the case of WebAssembly, that code is written to the in-script and API. Then when we want to build Google Earth for WebAssembly, we just compile targeting that architecture, instead of producing a Native Client binary, it produces a WebAssembly binary instead as a result output. The actuals and web application itself is all the same JavaScript that's built around it.

[0:29:47.2] JeM: Describe the state of tooling around WebAssembly today.

[0:29:53.4] JoM: It works. I mean –

[0:29:54.7] JeM: It works. All right.

[0:29:58.8] JoM: Yeah. There's some nuance to it though. The tool chain itself is coming together. I would say it's still probably a little bit early on for us. It does work. We can produce several versions of our application, both single and multi-threaded using WebAssembly. It works in all runs. However inside of that, there are limitations I think in some of the API, some of the tooling and how it works and I'll give you a specific example of that. In WebAssembly, there's basically two ways to do your compilation. They're called backends. One of them is the LLVM backend and the other one is the asm to asm backend.

Everyone is pushing in the WebAssembly world to start using the LLVM backend, however it's not the default environment, it doesn't work with Google Earth, we aren't able to use it. As a result, our compilation step is really to compile out and asm.js binary and then have it converted into a WebAssembly binary, or as a result, it makes build times longer, it doesn't support as many features. An example of that is something like SEMD, also source map debugging. We're using that today, but we'd love to move on to the LLVM backend, and I think the WebAssembly and the scripting community would like us to do that as well, or just everyone to do that, because it is supposed to become the standard way of doing it, but just not there yet, for example.

I would say that as far as being able to build an application in WebAssembly, it's actually not very hard. It's pretty easy to write to the API and then get something out of it. As far as little tiny nuances in through the whole flow that are still being optimized and worked out and fixed and so forth.

[0:31:30.3] JeM: There's some great writing by Lin Clark and Till Schneidereit and some other people from Mozilla. There's also a lot of good content for people from Fastly, who have written about the state of WebAssembly. This is really a big project. It's tremendous in scope in terms of how much tooling and compiler stuff needs to be built. Can you help us paint a picture, why is there so much tooling that needs to be built to enable this thing? You articulated by just saying, “Yeah, this thing just lets you run languages other than JavaScript on the web.” Okay, sounds fairly simple. Why is there so much work involved in making this thing a reality?

[0:32:14.2] JoM: Yeah, compilers are hard. I don't know. That's not the answer I know that I have a great answer for. I mean, I think there's just the amount of detail that goes into making

things work on so many varied amounts of hardware, on so many different browsers, etc., etc. It's just very complicated. Maybe a small example I can't think of that we've ran into is that when you want to write a WebAssembly binary, you actually write to what is called the unscripted API in C++. It provides all these methods on ways to do Network fetches, to starting threads, to all kinds of just things, right, the browsers do.

What we found is that as we ride against those methods and stuff, we run into all these little details. For example, one of the things that we run into pretty commonly is that we're really interested using multi-threaded WebAssembly binaries, because it really helps us with performance. However in that API, there's so much detail and so many methods that just weren't written with that in mind originally, that we do – we run into bugs a lot where it's like, “Oh, this method is blocking on the main thread and it really shouldn't be. Why is that?” Then, “Oh, okay. Well, we have to fix this, right?”

There's just so much I think detail in making every little bit work and work well, that it is really complicated and it takes a lot of time. The only thing too for us is a product that just wants to use WebAssembly, browsers have to do as much implementation as there is tooling that needs to be built. It's not so much that you can produce these binaries with the tool chain and the compiler and all that stuff. It's that the browsers need to then interpret that bytecode and actually run it accurately.

That's some place that we really struggle as just a product that wants to use WebAssembly, is because the support for WebAssembly across the ecosystem is pretty varied actually, and especially around the support of multi-threading.

[0:34:08.3] JeM: I'd like to go deeper into the subject of threading. What kinds of modules that you write in WebAssembly are multi-threaded? What do you need out of multi-threading support for Google Earth?

[0:34:24.2] JoM: Yeah, so Google Earth is a 3D graphical application, right? Basically what that means that it's hard is that we have this infinite loop that just runs on the main thread and tries to draw as many frames as fast as it can. Inside of that, there's a lot of just other work that we need to do that interrupts that main thread. Ideally, that main thread should loop as fast as

possible, do as little work as possible and just draw frames. That's what gives the user a smooth experience, right? Because Google Earth is constantly fetching things over the network, it's decompressing data, it's doing all kinds of other work, if we can't run that on background frames, that means we have to do it on that main thread and block.

The user gets a slower experience, the thing will freeze while it's trying to decompress that 3D model, so it then render it, etc., etc. Threading is actually pretty important for any graphically-intensive application. Without that, you just get jank and the slower experience for the user. Threading is really important for us in the sense that we can background so many things that the main thread doesn't need to worry about, and then just deliver the data to the main thread as it's ready, and let it pull it in and draw it very quickly.

[0:35:39.3] JeM: This may be a dumb question, or a question that doesn't really make sense, but how would this be different if it was just written in JavaScript? Just frontend JavaScript code that was executing in a single threaded fashion?

[0:35:57.9] JoM: Yeah. I mean, obviously JavaScript is a generally speaking, single-threaded in the sense that it's just one single thread and one overall runtime that you have to do stuff in. You could do that. I mean, with JavaScript you can use workers to do other work. If you look at the way WebAssembly is actually implemented, it's actually using web workers as their secondary threads. It's the same thing in a lot of ways. I think if Google Earth was written in just native JavaScript code, we probably try to use workers for a lot of the work that we use threads for in C++ and WebAssembly. I feel you could probably go after the same approach in the same way a lot of times. I just don't know how the performance would compare, because I have never tried to write a complete engine in JavaScript before.

[0:36:43.1] JeM: The web worker API. I don't know much about how that works. What multi-threading support do web workers give you? Is it fake parallelism, or is it genuinely multi-thread, stuff actually being executed in parallel?

[0:36:59.6] JoM: Yeah, my understanding is I don't know what workers, I close myself either, but my understanding is that actual just true web workers are on separate threads. I think that – I'm pretty sure that is true. They're also then limited in what they can and can't do. They can't

interact with the DOM and do things. They can't write the graphical context and so forth. That's the same I think even in the threading model of WebAssembly, but what they can do is network fetches, store things to disk, do some processing that is computational and not necessarily rendering-based. That's the thing that I think you use those for. That's the same thing that we use threads for inside of both WebAssembly, Native Client on Android and iOS, etc.

[0:37:40.2] JeM: In college, I took a class on Android. I think the one thing that I remember from that class is assuming I remember it correctly, is you never block on the UI thread, like at the main thread that's rendering the UI. I think what you just said there is that in web workers, you can't alter the DOM, I guess. Given that we're talking about a rendering engine here for Google Earth, can you tell me about what are the best practices for a multi-threaded manipulation of a user interface?

[0:38:12.0] JoM: Yeah. Well, I mean, in the case of Google Earth again, it's that whole frame loop, right? We have that loop that's just cycling and cycling and cycling. Basically, anything that we cannot do on that thread, but still allow it to draw is basically what we optimize. We basically look at – we look at that loop, we look at all the function calls that are happening, we look at the type of work that is being done and then anything that we can find that we could do on a different thread, we do it that way.

The way it works for us is we actually have this job system in our code base to where we can just say, hey, the user just flew the camera to this location. We need to request all of this data, decompress it, prepare it for rendering. Then we say hey, various jobs, go off and do that on background threads and then just deliver that data to the main thread, the main loop once it's ready. Then it has to do as little as possible just to put it on the screen basically.

[0:39:10.8] JeM: That main thread, it never gets blocked. It's just grabbing data and then rendering that data.

[0:39:18.6] JoM: Well, I mean, that's not true. You optimize it as best as possible. It inevitably has to do some things. That's one of the interesting parts about these various platforms and their implementations that there are things they can and can't do off the main thread. While we do our best to optimize it, it's not perfect, right? You just do the best you possibly can.

[0:39:37.4] JeM: Oh, so when you mean various platforms, you're talking about iOS, Android, web, different browsers.

[0:39:42.4] JoM: It's not necessarily different browsers. I think the standards are pretty similar across browser. Yeah, definitely in different platforms. It's funny that sometimes the rendering thread will actually be different than the application main thread and sometimes they'll be the same. I can't quote you a platform-to-platform, which is which, but we have to deal with that in different ways on different platforms to do the best for performance.

[0:40:05.8] JeM: It sounds like your adoption of WebAssembly today, a lot of it is you're porting native client code to web assemblies. Is that what you said, or are you writing brand new modules in WebAssembly also?

[0:40:19.4] JoM: No. Our whole entire code base since it's so cross-platform, we've extracted away, the specifics of one platform to another, to a very extensive degree. The amount of Native Client specific code and our raw code base is super tiny. The amount of asm and scripted specific code NR is also super tiny, and the same goes for iOS and Android. Really, we just basically do our best effort of implementing to that API as close as possible with whatever they're providing that API. Sometimes there aren't equivalents.

The basics are a robust networking framework that we can write to, because we need to be able to make a bunch of network calls and then handling user input events is another big part of that layer, right? As far as touches and taps and swipes and zooms and whatever else. Other than that, I mean, it's not a lot of, if you will, platform-specific code. The rest of it is pretty abstracted. There's not a lot of code that we actually have to port and change. We just have to write it to whatever that particular platform is capable of at the time.

[0:41:23.7] JeM: I'm a little confused. What do you use WebAssembly for today? Can you give me a few examples for modules where you are using WebAssembly in the stack of Google Earth?

[0:41:38.6] JoM: Anytime a network request is made, that goes through WebAssembly-specific API calls. Our network stack at a large is it's abstracted, so we just inject class that applies to the interface and then it makes those calls, right? The other things that we do is yes, those user input events, right? The API and mappings for all of those are very platform-specific, so we write code that takes in that specific event, translates it into a generic model and then passes that into the application for processing.

All of the processing of these events responding to the response from a network is all the same. The only real difference is at the edges, right? The edges just do whatever is specific, manipulates that result into something that our common code just understands and passes it along. Someone who works on Google Earth, they don't spend a lot of time having to think too hard about what the platform specifically is.

We can just think about handling what happened, the event that happened and the network requests came back etc., etc. Now when you were writing the actual UI though, the UIs themselves are very platform-specific. That goes to our architecture a little bit, is that we have all this common C++ code where everything is abstracted as far away as possible and nobody really has to generally speaking worry about it, unless they're the ones maintaining that interface.

We do stop at the view, so we follow a design pattern called model-view-presenter. The view itself is implemented completely in the Android SDK, the iOS SDK and JavaScript. We use polymer as our main framework on the web. The reason we do that is so that we can take full advantage of the actual platform when it comes to the UI and the user interface, but everything else is very generalized, very generic, very cross-platform. All that [inaudible 0:43:29.1], state management, rendering etc.

[SPONSOR MESSAGE]

[0:43:40.3] JeM: DigitalOcean is a simple, developer-friendly cloud platform. DigitalOcean is optimized to make managing and scaling applications easy, with an intuitive API, multiple storage options, integrated firewalls, load balancers and more. With predictable pricing and

flexible configurations and world-class customer support, you'll get access to all the infrastructure services you need to grow.

DigitalOcean is simple. If you don't need the complexity of the complex cloud providers, try out DigitalOcean with their simple interface and their great customer support. Plus they've got 2,000 plus tutorials to help you stay up to date with the latest open source software and languages and frameworks.

You can get started on DigitalOcean for free at do.co/sedaily. One thing that makes DigitalOcean special is they're really interested in long-term developer productivity. I remember one particular example of this when I found a tutorial on DigitalOcean about how to get started on a different cloud provider. I thought that really stood for a sense of confidence and an attention to just getting developers off the ground faster. They've continued to do that with DigitalOcean today. All their services are easy to use and have simple interfaces.

Try it out at do.co/sedaily. That's do.co/sedaily. You will get started for free, with some free credits. Thanks to DigitalOcean for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:45:41.8] JeM: The networking calls for example, zooming in on that for the sake of illuminating us further, the networking stack that's written in C++, this is going to be a complicated area of a code base I would assume, because – well, or maybe not complicated, but it has to be carefully architected, has to be fast because you're doing as I'm rotating Google Earth, the networking stack is going to be fetching things from the backends, could be fetching these quadtrees and swapping. Then you're going to be throwing away old quadtrees that you don't need anymore, because you just rotated the earth. The networking requests could be pretty intensive, and so you need high performance out of that that networking stack. Am I articulating that correctly?

[0:46:26.9] JoM: Yeah. I guess. I mean, if you think about it, a networking interface is pretty simple though. I mean, and the platform is capable of what the platform is capable of. I mean, the biggest demands that we have, obviously besides ideally speed, right? Network latency is

not in the processing, it's actually it coming back from the network, right? Any platform a computer is going to be faster at handling the response, than necessarily the thing that's really slowing it down is actually fetching and transmitting the data.

[0:46:55.7] JeM: I guess, what I'm having trouble understanding is you have this Native Client thing that allows you to use C++ in the web or anywhere else. You're migrating that to WebAssembly. Can you remind me why you need to make that migration?

[0:47:11.7] JoM: Well, the one thing, Native Client is only available in Chrome. That's probably the biggest problem, right?

[0:47:16.4] JeM: Okay.

[0:47:17.2] JoM: Yeah. We want Google Earth to be available to as many people as possible. WebAssembly is the new way that browsers are adopting to do this.

[0:47:26.2] JeM: Can't I use Google Earth on iOS?

[0:47:29.2] JoM: You can, but it's an application you'd install from the store, right? That's native code compiled in a different way. Then an application ostensibly written in objective-C around it.

[0:47:40.9] JeM: Oh, okay. That is not using the same networking stack.

[0:47:44.2] JoM: Yeah, it's using the native networking stack that's available in the iOS SDK, right? It's just delegating it out to that. The same thing happens with Native Client and wasm for the web is that and at the end of the day, the browser was just making those requests fundamentally as Ajax requests.

[0:47:57.6] JeM: Okay, so is this about getting Google Earth on other browsers essentially? That's what the porting effort is about?

[0:48:05.2] JoM: Yeah, for us as a product, that's our biggest win is that we can get Google Earth to more people. We really wanted to bring Google Earth to the web, because most of its

14-year history, it's been a desktop application and a mobile application. We really wanted to bring it to the web and make it just available to more people. One of the interesting pieces of that is that when we launched Google Earth on the web, it was the first time the Google Earth was actually available to Chromebooks, if you think about it, because you can't install native applications on a Chromebook. I mean, I know they're adding Linux support, and so forth.

We really lost out in that whole area of the world that may want to use Google Earth. They didn't really have a great option, until we launched on the web. Being Chrome-only is a pretty big disadvantage for us, because it limits our audience, right? We want to meet people on the browsers they want to use with our web product and WebAssembly is that's what's going to give to us, is that Mozilla, Apple, Microsoft, all of these people are actively working on WebAssembly implementations. They never started to my knowledge, Native Client implementations.

[0:49:10.3] JeM: When was Google Earth first available on the browser?

[0:49:13.9] JoM: It was just over two years ago.

[0:49:15.9] JeM: Wow, okay.

[0:49:17.1] JoM: Two years ago, you got Google Earth in the browser, but it was only available on Chrome because you only had Native Client access there, or was it just degraded performance on Firefox?

[0:49:27.9] JoM: No. We never actually launched. We never launched on anything more than Chrome, because the performance was just so bad. Because at the time, the only real option we had besides using Native Client was probably asm.js. asm.js maybe as you know is literally compiling native code into JavaScript code that and then is just process and parse. One, it's also single-threaded. The binary sizes are huge as a result, because it's literally text JavaScript. It was just really, really slow. It did run another browser, but it was just really slow and a very terrible experience, so we never actually launched it to the world.

Then we had heard of WebAssembly at that point, because it has been around for a while as a growing standard, but it hadn't really been standardized to W3C yet and they hadn't actually

started very much on browser adoption. Even today, browser adoption is starting to come online, but it's very different per browser and so on and so forth. Even as we moved to WebAssembly, our hope is that it becomes a cross-browser application, but it's still going to take time because those browser manufacturers need to get their WebAssembly implementations up to par, if you will.

[0:50:35.9] JeM: Well, this is yet another interesting side of WebAssembly. We've done probably 10 or 12 shows about WebAssembly at this point. Every episode I'm like, “Oh, WebAssembly is like, it really adds a lot to the internet.” Do you have any other predictions for how WebAssembly will change our interaction with the internet?

[0:51:00.0] JoM: Well, I mean, I think its biggest hope is it probably opens up a lot of different technologies and libraries and so forth to the web, right? Because WebAssembly even allows you to compile. There's so many different languages that people have done ports that will compile out as WebAssembly these days. I can't even keep track of them. We use C++ which I guess is maybe the primary language, but people can do Rust. I think I've seen PHP implementations. There's just so many different languages and technologies that you could now bring to the web. I think for WebAssembly, that's probably the larger win.

The other thing I think a lot of people are really hoping for is that I think people tend to use native code in cases of higher performance needs than JavaScript can deliver. I've actually gone to a few of these WebAssembly W3C committee meetings and stuff. I'm not actually on the committee, but I tend to attend because we're just really interested in the technology. There are things, everything from banks and other stuff. You just have to do high-throughput transactional stuff, and they want to do it in a browser, and so they're really hoping WebAssembly gives them the performance that JavaScript doesn't.

[0:52:03.0] JeM: Wow. I mean, you just think about from the point of view of I want to write client-side browser code in Go, for example. I don’t want to use JavaScript to write my client-side browser application. I want to do it in Go. I want to do it in Java. I want to do it in whatever other language. That's pretty transformative.

[0:52:25.1] JoM: Yeah. Yeah, and I've seen a lot of experimental applications and so forth of all kinds of different languages that will output a WebAssembly. I don't even know all of them.

[0:52:35.7] JeM: You could work on many different engineering problems. What motivates you to work on Google Earth?

[0:52:43.6] JoM: Interesting. Yeah. I actually was a CS degree interested in computer science. I didn't start in geo and then come to technology. I actually started in technology and came to geo. What really changed it for me is that I got some projects that were related to doing some just some basic mapping and so forth. I just loved the use case so much. Everything happens somewhere. I can't honestly think of a technology-based use case, application use case that isn't somehow enriched, or enhanced by geospatial technology anymore. To be able to work on a product that's literally about helping people explore the world around them, I can't imagine working on anything else at this point.

[0:53:30.1] JeM: It is such a positive some technology, because we can imagine so many different applications that can be built as this technology reaches maturity, which it feels like it's – I mean, today Google Earth, it's one of these things like VR. I think coincidentally, Google Earth on VR is a pretty cool application. It's one of these things where it looks today it's like, "Oh, maybe a triviality. I can tour the earth and virtual tourism." The future applications for this thing, you can imagine lots of business applications and monitoring applications and things that help us do better climate science. Do you see Google Earth is potentially becoming a utility platform?

[0:54:18.5] JoM: Well, I mean, that's actually already happening today. That's been happening for years. I can't tell you how many crazy use cases. Give you some small ones just in business, like real estate agents use Google Earth a lot, because they want to look at property lines and understand the details about the thing that's under sale and stuff. Solar planners use Google Earth today as well. I actually got solar installed on my house a few years back and it was really funny, because it's one of the people there to give me the quote showed up and he literally got a laptop out, put it on my kitchen table and opened up Google Earth in front of me.

“While looking at the pitch of your roof, it seems the sun, that tree is not going to be a shade issue, blah, blah, blah, blah, blah.” It's crazy how many business cases there are already where Google Earth is an integral part. You talk about things climate science, humanities and stuff. Actually, I got my start before Google actually in crisis response. Their Google Earth was just indispensable, because it's –

[0:55:17.5] JeM: What kind of crises?

[0:55:18.9] JoM: I actually used to work for the US government. One of the things I responded to was actually the Haiti earthquake. Basically, I was just in geospatial information technology to where I was gathering data. My whole job was to bring all of this information together and project it onto Google Earth, such that people could get a common operational picture of what was going on. Because in that crisis, we had so much information, but no way to make sense of it, and Google Earth was the tool that everybody rallied around to bring it together.

[0:55:50.6] JeM: You are a tech lead manager. That is a very specific role. Can you tell me what that means? What does a tech lead even do and then who do you – what are you managing?

[0:56:01.7] JoM: Okay, so yeah, I don't know how common these titles and labels are for other companies, but basically what a tech lead manager means is that I contribute technically as a software engineer, but I also manage other engineers. That's the distinction. As a tech lead, I'm expected to guide our architecture and ensure best practices and so on and so forth. We have a series of tech leads in our overall engineering group, some of which are managers, some of which are – in my case, I actually manage the people who work directly on the web platform, as well as our older desktop install platform.

[0:56:37.9] JeM: Okay, final question. How do you see WebAssembly being useful outside the browser?

[0:56:44.5] JoM: To be honest, I haven't thought about it that much.

[0:56:47.5] JeM: Okay, that's a fair response.

[0:56:50.3] JoM: I'm so focused around getting our application working really well in a browser that, yeah.

[0:56:55.6] JeM: All right, well fair enough. Okay, one more question. Google Earth for VR, do you use that, or how cool is that? I haven't tried it.

[0:57:03.0] JoM: You should definitely try it. We have actually a setup in our office. It's actually based on the same technology that web, Android and iOS is based on, so it's very much a similar experience from the 3D rendering perspective. It's literally meant to let you just fly around the world in a reality environment. Its feature set, I would say it is much simpler. It's more just about flying around cool places and looking at things, but it's definitely an amazing experience and I totally recommend you try it out.

[0:57:33.0] JeM: All right, well good endorsement. Jordon, thanks for coming on the show. Great talking to you.

[0:57:36.2] JoM: All right, thank you.

[END OF INTERVIEW]

[0:57:41.0] JeM: Commercial open source software businesses build their business model around an open source software project. Software businesses built around open source software operate differently than those built around proprietary software.

The Open Core Summit is a conference for commercial open source software. If you are building a business around open source software, check out the Open Core Summit, September 19th and 20th at The Palace of Fine Arts in San Francisco. Go to opencoresummit.com to register.

At Open Core Summit, we'll discuss the engineering, business strategy and investment landscape of commercial open source software businesses. Speakers will include people from HashiCorp, GitLab, Confluent, MongoDB and Docker. I will be emceeing the event and I'm hoping to do some onstage podcast-styled dialogues.

I am excited about the Open Core Summit, because open source software is the future. Most businesses don't gain that much by having their software be proprietary. As it becomes easier to build secure software, there will be even fewer reasons not to open source your code.

I love commercial open source businesses, because there are so many interesting technical problems. You've got governance issues. You got a strange business model. I'm looking forward to exploring these curiosities at the Open Core Summit and I hope to see you there. If you want to attend, check out opencoresummit.com. The conference is September 19th and 20th in San Francisco.

Open source is changing the world of software and it's changing the world that we live in. Check out the Open Core Summit by going to opencoresummit.com.

[END]