Code Together Podcast
Episode 7: Let Us Entertain You
Host: Nicole Huesman, Intel
Guests: Krystian Ligenza, Autodesk; Michael Voss, Intel

Nicole Huesman: Welcome to *Code Together*, an interview series exploring the possibilities of cross-architecture development with those who live it. I'm your host, Nicole Huesman.

Autodesk makes software for people who make things. The company is best known for its architecture, engineering, and construction software. Its leadership in media and entertainment may not be as well recognized outside the visual effects industry. However, top VFX and 3D animation studios around the globe have relied on Autodesk software to produce countless award-winning movies—movies like Sony Pictures Animations' *Spiderman: Into the Spiderverse*, Disney's *Frozen* and Laika's *Missing Link*. Leading game developers use the company's software to create many of today's hottest games.

I'm excited to welcome Krystian Ligenza to this episode of *Code Together*. A self-proclaimed passionate troublemaker, Krystian works as a software architect on Autodesk Maya. He's also spent time as a software engineer on Autodesk MotionBuilder's development team. Hi Krystian.

Kyrstian Ligenza: Hi Nicole. Hi everyone. Great to be here.

Nicole Huesman: Mike Voss also joins us today. Mike is the original architect of the Threading Building Blocks flow graph API, a C++ API for expressing dependency streaming and data flow applications. He's also co-author of over 40 publications on parallel programming topics, including the recently released book *Pro TBB: C++ Parallel Programming with Threading Building Blocks*. Great to have you with us, Mike.

Mike Voss: Hi Nicole. Hi Krystian. It's good to be here.

Nicole Huesman: Krystian, can you tell us more about Autodesk's role in the media and entertainment industry and what you do at the company?

Kyrstian Ligenza: Yes, of course. You actually started with the hardest question at the very beginning 'cause lots of people know us from pretty pictures, TV series, movies that they watch many, many, many times or games that they were playing. But in fact, what we do, we have a collection of tools that animation pipelines can use to produce stunning visual effects, rendering, character animation. Our core products are Maya, 3D Studio Max, Arnold, MotionBuilder and Mudbox.

I joined Autodesk around 12 years ago as a contractor for MotionBuilder. MotionBuilder is our 3D character animation package; it's built like a game engine, but allows you to capture, edit live performance and it's optimized for animation and directors. So one of the movies that I often bring up when I explain what MotionBuilder is for is *Avatar*. This is the movie where MotionBuilder was used during the previsualization stage and the director was exploring how to tell the story in the best way before the production started and leveraged all the real-time capabilities of MotionBuilder. So that's how I started at Autodesk. After that, I, of course, relocated to Canada and became a permanent employee. In 2013, I took an opportunity to leverage my knowledge and MotionBuilder experience to solve some of the hardest problems we have in Maya. Maya is our 3D computer animation software that also enables artists to do modeling, simulation and rendering. These days, besides animation performance, which is the main thing I was doing with Maya, I'm also involved in open source collaboration and adoption of open standards.

Nicole Huesman: Excellent. Thanks Krystian. As consumers, our hunger for increasingly immersive movies and games only continues to grow, which leaves animators and game developers in a continuous race to keep up with these demands. Mike, how has Intel Threading Building Blocks, or TBB, evolved to help them?

Mike Voss: So just like Krystian described the tools that Autodesk provides as tools to help artists and animators fulfill their vision, what TBB is for is to help developers like Krystian to create those and to harness the power of Intel hardware. And so, Threading Building Blocks has a long history. It was first released back in 2006, just at the time that multi-core processors were just becoming available. And one of the selling points back then was instead of developers worrying about all the low-level details of how to take advantage of multi-core through threads and low-level features, they could use TBB and therefore be future-proofed. And by future-proofing, we meant that we knew that more cores would become available in the future. We knew that hardware would become more complex. So by targeting TBB, developers could focus on what was important to them—their tools—and just express the work they wanted to schedule onto the hardware, and TBB would do that for them.

So over time, hardware has become more complicated and applications have become more complicated and TBB has evolved to support hardware as more cores became available, as heterogeneous computing became something that was important, and also to help application developers manage as their parallelism became more complicated, as they have nested parallelism, as they were supporting third-party plugins that had parallelism, and so on. And so that is what TBB's role has been in helping developers like those at Autodesk manage the increasing complexity of both hardware and software over time.

Nicole Huesman: So as you mentioned, there's this diversity of hardware, the rise of heterogeneity, the widespread use of accelerators. Krystian, can you talk about your journey in accelerating Maya and how you're tackling this whole concept of heterogeneous computing?

Kyrstian Ligenza: Of course. Like Mike mentioned, hardware is becoming more and more complex and can handle more and more data. Similarly, artists need to deal with more complexity and generally we're targeting bigger and bigger characters that are becoming more and more realistic. This creates a need to leverage all available resources, including CPU, GPU threads, but as well as CPU and GPU memory. Maya is used for modeling, animation, simulation, rendering, like I mentioned. An entire computation is authored with a single extendable authoring graph. By extendable I mean, customers can write and are writing many, many plugins on top of it.

Each domain—so, modeling animation—has different performance bottlenecks and different workflows. So, for example, if I take simulation, which you can see if you watch *Avatar*, you can see characters having some animated, simulated clothes, or you can see *Avengers* and you see Thanos basically looking like a real character. Simulation typically happens in graphs that are relatively simple. There are just a few computation nodes, but the computation takes quite a lot of time. Looking at a different domain—character animation—so, for example, Disney's *Frozen*, you can see characters running around—they're a simulation as well, but the main animation is driven by rigs and animators just tweaking quite a lot of parameters to pose this character and tell the story. And it's a body or it's a character's facial expression, all of that complexity for character animation and the workloads that they have are completely different from simulation. Typically an authoring graph in this case will have on average, around 15,000 nodes to describe computation. And these nodes are very different. Some of them are running very fast, they're lower than one millisecond. Other deformers that tend to be at the end of the computation for a single refresh frame can take over milliseconds. And these nodes typically are maybe a bit similar to simulation, meaning they can leverage a parallel algorithm within this execution.

So as you can see, we have different domains, each having different needs, and we have to use all the available resources. We kind of categorize them in a way. Characters we know those single nodes, less than one millisecond. They are good on CPU. We will want to schedule them on CPU. The deformers, they're very good to be run on GPU, especially when very often they end up being dropped by the viewpoint render. And then simulation you just want to have massive amounts of tracks and be able to express the algorithm in a way to schedule it. So these different workloads and the need to leverage available resources to handle the complexity was the challenge that we needed to solve on Maya. And that's where I joined Maya. From past experience we learned that node level parallelism is not enough. So just doing a parallel for or a parallel reduce within a single compute wasn't enough. We also proved that just doing the data files and from the consumer side, like viewport is not capable of understanding all the dependencies that a dispatch or a scheduler needs to have in order to coordinate computation between CPU and GPU.

We needed to do bigger transformations on Maya side, and to not go into too many details, I will just describe a big shift that we did. The first one was we decoupled evaluation out of rendering, meaning that we have complete control over what needs to be computed. We have one way of describing it. The second thing that we did was we took compute out of authoring, meaning that we can target with different evaluators, different hardware, or applied different optimizations without having to change how artists created their graph. It was the same description but we can target different hardware, apply different optimization. And at the very end, we also took complexity out of execution to be able to deliver even better performance. So the end result of all of that, there's a framework that we call evaluation manager, which takes advantage of any authoring graph. And by applying special evaluators can execute this graph on CPU or GPU, and leverage GPU memory to have cached animation for basically the next level of performance.

Nicole Huesman: Those are such, you know, really significant advances and from a consumer's perspective, it all leads to bringing together these award-winning movies, hot games, but it really lends to these fantastic and amazing entertainment experiences. So thank you. That's fantastic.

Kyrstian Ligenza: That's exactly the feedback we are getting from our customers. The big thing for them is they see the value without needing to change how they are creating things. Of course, there is a way to get even better performance if they apply certain optimizations and be able to start thinking in this parallel way or this tasked base way, like TBB is designed for. Now, customers are starting to ask different questions, saying what type of hardware should we be buying to get the best performance? Which, as we know, it's not the easiest, there are many dependencies, but we completely shift from being single-threaded application to multi-threaded and being able to target CPU and GPU and apply any optimization.

Nicole Huesman: Mike, let's bring you into this. Intel and Autodesk have been collaborating for so many years to really deliver these immersive entertainment experiences. When you've talked earlier about scheduling applications onto this complex hardware and considerations like non-uniform memory access or NUMA awareness. Can you talk a little bit about those considerations and your collaboration with Autodesk over the years?

Mike Voss: Yeah, sure. So we've taken this journey that Krystian's talked about. We took that with Autodesk and with other customers as well. So early on when TBB was introduced, it was all about that first step of just getting some threading into applications that had been sequential. So that was enabled through TBB.

Over time as hardware became more complicated, it was how do you introduce even more parallelism and, as Krystian talked about, some of that is some of your parallelism has really large granularity, some has fine granularity. How do you manage those things? And TBB evolved to assist with that as well. As accelerators became important, things that we started focusing on in TBB were how do we enable people to coordinate both computation on the host and offload to accelerators? So we introduced things in the flow graph like async node. In our basic tasking, we introduced resumable tasks. And as hardware on the host side becomes more complicated, we have to deal with, just as Krystian said, their users can get more performance if they change things a little bit. We added some hooks where if you didn't want to just hand things to TBB and let TBB try to deal with everything, you could express some hints to TBB that would enable it to do a better job. And some of those hints were around, for example, NUMA. So you could create groups now of work, where you would say, I want this group of work scheduled on one of my NUMA nodes and this other group of work scheduled on a different NUMA node, because if that happens, performance would be better. So this journey that Autodesk—as well as, you know, many of our customers has gone on—this journey has led to features that TBB has added over the years. And now with the introduction of oneAPI, we're really embracing this heterogeneous environment where TBB, now named oneTBB as part of oneAPI, is a part of that story and is the way that we can leverage over a decade of experience on the host to get good performance when you're targeting CPUs and also fit in underneath some of the kernels, like from our math kernel library, when they execute on the host, and also leverage that experience with coordinating on accelerators. So Intel has gone along the same journey that Krystian just described with our software products that enable our developers.

Kyrstian Ligenza: And I can add a few more examples, how we use flow graph and TBB. So, yes, Mike, flow graph was something that we leveraged at the very, very beginning of all the transformations that I describe. We had already past experience of some failures with getting this complex architecture to get to multi-threaded, basically, and we needed to derisk a new approach. When we started this work in three weeks, with TBB flow graph, and basically a single node from the flow graph continue node—basically, just a single input, simple output, and the broadcast to all successors—we are able to prove that we can take 22 characters—so, you can imagine any complex 3D character animation that has 22 characters in it—and we were able to convert it from all characters, executed sequentially to every single character being executed concurrently. That was massive. So that was the first example of how flow graph enabled us to prototype and derisk this transformation that we needed to do with Maya. As well, you mentioned the nested parallelism. That is something that, for example, in MotionBuilder, we never needed that. But in Maya, because of the domains that we are having that I was describing, simulation having different workloads than animation. Especially when you look at movies like *Avengers*, you see that these two have to be together in order to have a believable character, right? So Harris needs to move. Klaus needs to move. And animation needs to be believable. We need to be able to schedule not only at the graph level, but as well at the node level. So this nested parallelism was crucial for us in order to be able to build a real performance, not to mention just to balance execution. In this case, our artists, they don't necessarily want to learn how things are working, right? They just want to create something, model this compute and have it run as fast as possible. So this balanced execution that TBB allows is what we need for our artists.

Nicole Huesman: So in an earlier episode, we talked about the industry shift to C++, and the increasing capabilities to support parallelism. Krystian how does this impact the need for, and the use of, TBB at Autodesk?

Kyrstian Ligenza: So Maya is a large code base and we have homegrown solution to things like mutexes. We have some atomics that were written a long time ago when we were first doing, we're switching to TBB. Now, what we are realizing is modern C++ is providing some of those. So we are switching from TBB atomics to standard library atomics. Some mutexes we are replacing now with standard libraries. The same thing for thread which has almost exactly the same interface. So we are making the switch. We are able to build Maya with C++ 17. So we are trying to follow and be aware of what's happening. Of course, the scheduling that we described before, it's not yet there. So we are monitoring and we are very interested in knowing what will be the future and how modern C++ will play with TBB scheduler onward, especially for balancing. We just said, balancing, we have tasks and plugins. We have customers using whatever there is. We just want to see how it will play and how we can together solve this problem, so customers don't run into trouble.

Nicole Huesman: And I know Mike, you look a lot at the C++ standard as well and how it relates to TBB. Can you talk a little bit about?

Mike Voss: Yes. So TBB is a C ++ library. It's built using ISO standard C++, and over a decade ago, when it started, there were many fundamental features missing from the C++ standard like threads and atomics and mutexes. And as those things were introduced, we could take advantage of those in TBB. Initially we gave migration paths for customers. We knew that things were coming up in the standard, but in industry people can't always migrate to the latest compilers as fast as the features become available in the standard, so we would implement upcoming features using older versions of C++, so that there'd be a migration path. And so we're always evaluating what is the value that TBB adds on top of standard C++, and as Krystian has pointed out, one of those things now is that there are parallelism features increasingly available in C++, but oftentimes they describe that something will be parallel. Like there are parallel algorithms. But the standard doesn't define a scheduler that will actually implement that parallelism. And so over time, in fact, Intel provides a parallel STL—so, an implementation of those parallel algorithms—and those are on top of TBB as the execution engine. So we're monitoring C++, we're involved in the C++ standards committees. So we're always tracking and looking towards the future and seeing how TBB should evolve with C++, and what TBB's future will be in the context of future C++. So some of the upcoming things that people are working on now are things that are planned, hopefully for C++ 23 would be things like executors and senders and schedulers. And these, just like the algorithms, they define an abstract interface, but you need to plug in some vendor scheduler underneath. And that's where we see, again, TBB providing that execution engine for these standard algorithms for executors, for senders, et cetera.

Nicole Huesman: Let's shift a little bit and look at what's next. So, Krystian can you give us a glimpse into the future of Autodesk products and how they'll leverage increasingly heterogeneous systems?

Kyrstian Ligenza: From my perspective, you saw we made a couple of transformations in Maya. There's one more transformation that I think we have to do, and it's shown up already in this conversation where we said, if artists will change a bit, how she is doing things, she will get better performance. And I think that's the next problem that we have to solve together. Artists should not have to think about performance when they are working. They should just focus on the artistic aspects of their work, and let the software and hardware deal with making sure that it draws as fast as possible, leveraging all the resources that the artist has in their hardware. We describe it as decoupling performance from authoring. So basically when artists work and model compute, they don't think about performance.

Let's take an example. When an animator is working on a character, this character has a shape. This shape is driven by deformers, like I said before. It can be a single shape or it can be multiple shapes. Now we know that when there are more shapes, these characters are faster to compute on CPU, but that already constrains the entire production pipeline because the model there needs to think about generating multiple shapes instead of a single shape. And sometimes it's easier. That's one example.

Other examples are, in order to target GPU, the character needs to be built in a way that the leaves of the computation happen on the GPU. So let's say entire body is just driven by animation and then happens the deformation that is on GPU. Or it can happen that there is something that needs to stay on CPU just after this deformation and currently this is a problem because of the hardware limitations. We are not blocking CPU to wait on GPU. We are not synchronizing the two together because that's costly. So that's another one that we would like to have solution for. So artists, they just build it the way they have to build it. The characters behave as they wish but run as fast as possible and leverages all of the available resources.

There's also an aspect to it which connects to just-in-time compilation. So right now artists always work in environment, which is the authoring environment, meaning they can change anything, but at run time when they actually do the final animation, so when they just tweak those curves to have this motion, to tell the story at that time, they won’t change everything in these characters. So we can actually reduce those 15,000 nodes. On the extreme case, we heard 325,000 nodes, compress it into smaller number of executions, and then just have smaller number of TBB tasks at the end created. So I think that decouple performance from authoring. This is one thing. Another thing I think is just adoption of open standards, right? You mentioned oneTBB. There's many more. I think that's another future for all these products.

Nicole Huesman: And Mike, when you look into your crystal ball for Threading Building Blocks, what are you most looking forward to?

Mike Voss: So I think the future for TBB is bright as part of oneAPI, now called oneTBB. oneAPI is helping people more easily express applications for heterogeneous platforms, partly through a language called DPC++, which is C++ plus SYCL plus some Intel extensions. We have open standards for DPC++, and you can target any of your hosts or accelerators through DPC++. And why it's exciting for TBB is when you target the host, we can leverage that over a decade of experience, getting performance on the host, by executing using TBB as the engine. When you're targeting an accelerator, while TBB is not something that you would execute on a GPU, for example, you can still target the GPU and get a different type of execution engine on that GPU. So we can leverage that experience we have in getting performance on the host by using TBB underneath DPC++. And I think looking at standards like C++, a very similar thing holds there also. As abstractions like executors and schedulers and senders and the parallel algorithms are added to the standard, similarly, they need something to execute on top of, and again, TBB can fit the bill there as well. So I think while we need to be constantly aware of what the C++ and other standards offer, we don't need to compete with those. In fact, we encourage things to go into the standard. That is great for us because it enables the most developers for our hardware. That's something we're really happy to see when something gets into the standard. So our role is simply to look at how we add value on top of those standards and how we can influence those standards to best help our customers.

Nicole Huesman: And Mike, as we wrap up here, for those interested in learning more about oneTBB or oneAPI, where can they go to learn more?

Mike Voss: So for oneAPI, they can go to oneAPI.com to get information. For TBB, TBB is both available as a product but also as open source. So you can go straight to github to get information on TBB. You can download the source for TBB. You can build it yourself, or you can get prebuilt versions of TBB. And also I gotta plug, there is a new TBB book, *Pro TBB*, and that is something that you can download for free as a digital copy through APress.

Nicole Huesman: And Krystian, how can listeners learn more about Autodesk in the media and entertainment industry?

Kyrstian Ligenza: Like I said, at the very beginning, when you asked the first question, everyone knows us by the pictures that our customers are generating. I think I'll propose to first go to YouTube and just look for "Autodesk 2020 Film and TV, VFX Games and Design Showreel". It's a 4-minute video and you will see all the cool stuff that our customers are doing with our software. I think that's the first place to start with. After that, when you want to learn a bit more on the transformation that we did in Maya, a bit higher-level overview happened during SIGGRAPH 2017 and that's our talk, "Parallel Evaluation of Animated Maya Characters". And then when you want to learn even more details, I would encourage you to Google "Using Parallel Maya" white paper, and it describes all the work that we did in details and challenges that customers may have in order to get the best performance. So. I would encourage doing that. And of course, then the next level is learning all about TBB. So Mike just provided amazing info.

Nicole Huesman: Excellent. Krystian. It has been so great to have you on today's program.

Kyrstian Ligenza: Thanks for having me; it was great.

Nicole Huesman: And Mike thanks so much for sharing your insights with us.

Mike Voss: Thank you.

Nicole Huesman: For all of you listening. Thank you for joining us. Let's continue the conversation@oneapi.com. Until next time.

Code Together Podcast
Episode 7: Let Us Entertain You
Host: Nicole Huesman, Intel
Guests: Krystian Ligenza, Autodesk; Michael Voss, Intel

To learn more:

oneAPI.com
oneAPI Threading Building Blocks (on GitHub)
Intel oneAPI Threading Building Blocks
Autodesk 2020 Film and TV, VFX, Games, and Design Showreel
Parallel Evaluation of Animated Maya Characters (SIGGRAPH 2017)
Parallel Maya White Paper