WEBVTT

00:00:00.000 --> 00:00:00.000
So I I have close captioning turned on. I gonna go ahead and turn boarding on, and I will get ready to open up the webinar.

00:00:00.000 --> 00:00:09.000
Now

00:00:09.000 --> 00:00:14.000
Where's the camera it's actually in the far it's in the back.

00:00:14.000 --> 00:00:21.000
Yeah, your point got it. My head is chopped off.

00:00:21.000 --> 00:00:33.000
But it's not that important, anyway. Good talking Oh, there we go ain't no worry about it.

00:00:33.000 --> 00:00:36.000
Somebody just fixed it. Okay, We're live whenever you're ready to start.

00:00:36.000 --> 00:00:41.000
Yeah, Thanks. just making sure that the participants have leveled off.

00:00:41.000 --> 00:00:46.000
Gotcha. Yeah, it looks looks good. okay, hi everyone once again I'm.

00:00:46.000 --> 00:00:52.000
I'm Michael Litman the division director in size for the division of information and intelligent systems at some point.

00:00:52.000 --> 00:01:05.000
I'm going to stop stumbling over that i'm we've had now 3 this will be our third distinguished lecture that we've been running this year, and so far i've introduced all the speakers but that will

00:01:05.000 --> 00:01:08.000
stop today. This is this is my last one, and and other division.

00:01:08.000 --> 00:01:14.000
Directors in the directorate are going to be handling the future talks, but i'm super excited to get to do this one.

00:01:14.000 --> 00:01:24.000

This talk has been a long time coming. Pat Hanrahan was actually invited to speak as a distinguished lecturer by my predecessor before the pandemic, and he agreed to come.

00:01:24.000 --> 00:01:33.000
But then the pandemic happened, and it would. He was given the option of speaking to everyone virtually, but he really wanted to come in person.

00:01:33.000 --> 00:01:38.000
And so, when things restrictions started to lift, he reached out to us again and said, Hey, can I come in person?

00:01:38.000 --> 00:01:49.000
And we were so excited to set things up for him to do that a Pat Henrahan is a professor at Stanford and a recent touring award winner and he's won 3 accounts awards so that basically

00:01:49.000 --> 00:02:04.000
puts him halfway between Donald knuth and meryl streep pat works in computer graphics which is just fascinating area of computer science that has overlap with basically every other part of computer science it's got math

00:02:04.000 --> 00:02:08.000
and theory it's got systems it's got an aesthetic component to it.

00:02:08.000 --> 00:02:12.000
A design part. How you think about how people perceive things.

00:02:12.000 --> 00:02:21.000
Physics simulation. It really covers absolutely everything. and and Pat has been very, very influential, and his he's recently moved it.

00:02:21.000 --> 00:02:26.000
The field in the direction of programming languages and I think that's going to be part of what is covered in today's talk.

00:02:26.000 --> 00:02:32.000
Now I Hadn't met Pat before today but I would have known him in another life, because when I started my PHD.

00:02:32.000 --> 00:02:37.000
I worked with Andrew Wittgen at Carnegie, Mellon.

00:02:37.000 --> 00:02:40.000
Andrew went on to work at pixar on the renderman system, which was Pat's brainchild.

00:02:40.000 --> 00:02:49.000
So I assume that they've interacted quite a bit I've heard wonderful things about Pat from Daniel Daniel Ritchie, who's a junior colleague of mine at Brown, and one of

00:02:49.000 --> 00:02:58.000
Pat's former students Now, if you've not heard of pat or renderman, for that matter, you undoubtedly have seen what he and it have done.

00:02:58.000 --> 00:03:05.000
Renderman is the computer graphic system that turns descriptions of scenes into the amazing visual worlds you see in, for example, Pixar movies.

00:03:05.000 --> 00:03:11.000
So I am absolutely sure that you will find this talk both intellectually and aesthetically stimulating.

00:03:11.000 --> 00:03:21.000
So let's welcome, pat Hanraham, thank you thank you Michael That's very nice introduction, and it's great to be here, as I said.

00:03:21.000 --> 00:03:26.000
I sort of held out to have an in-person talk, so I know not everybody's able to attend.

00:03:26.000 --> 00:03:36.000
But I was excited to come and You know I just hope I get a chance to talk to some people in person afterwards as well, and for all those is still in virtual land.

00:03:36.000 --> 00:03:45.000
It's good to have you here as Well, so this talk is actually based on my Twitter word lecture, which also I just recently gave because I wanted to do that in person.

00:03:45.000 --> 00:03:50.000
And there's a lot of stuff here about my life as a scientist.

00:03:50.000 --> 00:03:59.000
Oh, well, first actually, the first thing I forget what I want to say here is I want to thank Nsf for all the support that I've received through the years.

00:03:59.000 --> 00:04:08.000
When I first went to Stanford I got a large, a medium scale infrastructure grant which actually launched the computer Graphics laboratory at Stafford.

00:04:08.000 --> 00:04:15.000

It was a five-year grant fairly large Grant and that just you know, Stanford really didn't have computer graphics.

00:04:15.000 --> 00:04:21.000
And without that Grant we went to Bill to set up our program, and I think it was a very successful and still is a very successful program.

00:04:21.000 --> 00:04:27.000
And then, of course, I have many had many individual words throughout the years as well.

00:04:27.000 --> 00:04:30.000
I, in my last real, significant interaction with Nsf.

00:04:30.000 --> 00:04:34.000
Was we working with Larry Rosen bloom on the Fedava program?

00:04:34.000 --> 00:04:40.000
So I was in his sort of core. in her team. We had a workshop and let me launch that program which was on data analysis and visualization.

00:04:40.000 --> 00:04:45.000
So, you know, Andsf is really dear to my heart and I want to thank you for my research.

00:04:45.000 --> 00:04:51.000
I won't be even here to build funding my research I wouldn't be here if you hadn't done that.

00:04:51.000 --> 00:04:57.000
So. thank you. so, anyway. But this talk is a bit of a retrospective on my career, and I want to start.

00:04:57.000 --> 00:05:04.000
Actually, I added, this slide about my advisor. Okay, So I actually am not a computer scientist.

00:05:04.000 --> 00:05:07.000
I've never really taken any computer science courses in my life.

00:05:07.000 --> 00:05:20.000
I was trained as a physicist, and the person on my left was my undergraduate Advisor Heinz Barshall, who worked on the Manhattan project, and he studied basically cross-sections of

00:05:20.000 --> 00:05:32.000
various atomic reactions, like in particular neutron cross-section, said he measured what the probability of not neutron being absorbed into uranium and cause and fission. was and he was a great experimentalist.

00:05:32.000 --> 00:05:40.000
Actually, and I actually learned that analysis from him, too. I mean, he obviously, you know, as an experimentalist you have to do that analysis.

00:05:40.000 --> 00:05:52.000
But he was always interested in how we could confirm theories with experiment, and I learned physics from him, and he was an unbelievable mentor as a even as an undergraduate My other.

00:05:52.000 --> 00:05:59.000
2, I decided to leave sort of more theoretical physics, particle physics, and go into biophysics.

00:05:59.000 --> 00:06:02.000
And so I had 2 advisors there, and in my PHD.

00:06:02.000 --> 00:06:07.000
One was Leonard Year which i'll talk about he actually studied a deep learning.

00:06:07.000 --> 00:06:19.000
This is in 1,980, purely calling pyramidal neural networks, and in fact, he was about the only guy around still studying neural networks, and I was in a neurobiology at the

00:06:19.000 --> 00:06:23.000
time, and of course I was super into neural networks.

00:06:23.000 --> 00:06:32.000
But you know that was considered really bad bad idea to be in neural networks. So i'm still trying to reconcile that with the modern world.

00:06:32.000 --> 00:06:43.000
And then, and then my advisor, my main police advisor, was Tony Strand, who was in the department of zoology and studied neurobiology of of invertebrates.

00:06:43.000 --> 00:06:52.000
So I worked on the motor nervous system of Ascarus lumberquitis, which is a neat and Tony.

00:06:52.000 --> 00:06:56.000
My advisor. I came from Sydney banner's lab who worked.

00:06:56.000 --> 00:07:10.000
He started to work on C. Elegans, and you know what I really learned from Tony and and his extended family was, which includes Cindy Brenner, Francis Crick, John Zulston, who ran the Human Genome

00:07:10.000 --> 00:07:20.000
project and won a Nobel prize for programmed cell taposis was to do long-term research projects.

00:07:20.000 --> 00:07:26.000
And I think this project i'm going to talk about it was actually like my 30 year Long-term Research project.

00:07:26.000 --> 00:07:37.000
But you know I was just very fortunate in in in my graduate career to have advisors who stressed long-term research, not just writing individual papers.

00:07:37.000 --> 00:07:41.000
And and so you'll see that in the talk fight get to it.

00:07:41.000 --> 00:07:50.000
Okay, but anyways, I really want to thank them and i'll have more to say about my career. Maybe a little bit. I don't want to elaborate too much time in, because I almost want to talk about technical stuff, but I

00:07:50.000 --> 00:07:56.000
do think you'll see there's a sort of trend towards being us in long-term research.

00:07:56.000 --> 00:07:59.000
And in my career. Okay, So let me start with the real stuff.

00:07:59.000 --> 00:08:04.000
So I went before I got the Pixar. They made this picture, which is a very famous picture in computer graphics.

00:08:04.000 --> 00:08:12.000
This is ancient history, called the road to Point. raise but point and raise is a national siege point. raise national seashore. It's just north of San Francisco.

00:08:12.000 --> 00:08:21.000
It's a very beautiful place, and they were working for Lucas film, and they were trying to convince him that computers could be used to make imagery for the movies.

00:08:21.000 --> 00:08:30.000
So they made this as a teaser for him and the idea was, If you're George Lucas hello like Hot No, the Hot Rod sports car, you know.

00:08:30.000 --> 00:08:35.000

He would be driving along this road, and you know he just imagined him like driving into Porn Range.

00:08:35.000 --> 00:08:45.000
What would he see, you know, like, and suppose you wanted to create this world that he and he might imagine you drive something like that. And he'd say, I want to put that in star wars, or something like that so you just had to be able to do a

00:08:45.000 --> 00:08:49.000
bunch of things, and this was just trying to demonstrate where they had got.

00:08:49.000 --> 00:08:52.000
So you had to be able to make, you know, terrain like mountains.

00:08:52.000 --> 00:08:58.000
You had to be able to make. waves like waterways. you had to do atmospheric scattering like clouds and rainbows.

00:08:58.000 --> 00:09:02.000
You had to be able to draw, you know plants or make plants.

00:09:02.000 --> 00:09:04.000
You know, man-made objects like fences, and so on.

00:09:04.000 --> 00:09:07.000
And the idea was I just what would it take to do this?

00:09:07.000 --> 00:09:17.000
And trying to show him you could do it. And then the the system we had at Pixar was called Rays, which actually the name was invested by Lauren.

00:09:17.000 --> 00:09:23.000
Carpenter for renders everything you ever saw or could see.

00:09:23.000 --> 00:09:29.000
Okay. And so the idea was like, So what would it take to build a system that would renders anything you could see like in the everyday world?

00:09:29.000 --> 00:09:37.000
And that was our motivation. So that was our problem statement in some sense like, what would it take to do that? Ok?

00:09:37.000 --> 00:09:48.000

Very ambitious. And then there was one other thing that turned out to be important which led to this idea of photorealism, which is that you had.

00:09:48.000 --> 00:09:56.000
You take live action. that was taken with a camera and you have to be able to overlay computer generated stuff, and they'd have to merge seamlessly.

00:09:56.000 --> 00:10:00.000
Okay, that is the the real world and the virtual world.

00:10:00.000 --> 00:10:02.000
Had you come together like sort of augmented reality?

00:10:02.000 --> 00:10:04.000
And sometimes can you augment the real world with this virtual thing?

00:10:04.000 --> 00:10:11.000
And this is an example of an early thing. They did from this movie. young Sherlock Holmes. were the stained glass man on family.

00:10:11.000 --> 00:10:20.000
I've seen this but sting last man so jumped off this stained glass window, and now walking around in the church, and that was really hard, because you could not have any artifacts.

00:10:20.000 --> 00:10:25.000
You know you couldn't have like jagies you couldn't have any Alicine artifacts you also had.

00:10:25.000 --> 00:10:32.000
It's surprisingly, hard to simulate a real camera because you have to have depth of field and motion blur and other things like that.

00:10:32.000 --> 00:10:34.000
And you, of course, have to do the lighting correctly.

00:10:34.000 --> 00:10:38.000
So you have to inherit the lighting from the real world on the virtual worlds.

00:10:38.000 --> 00:10:43.000
You have to have some interaction there. So this was actually a hard thing to do.

00:10:43.000 --> 00:10:54.000
And then the other really hard thing about it. And this this slide came from a retreat that I actually led, because we wanted to build a raise machine right as a machine that could render everything you ever saw.

00:10:54.000 --> 00:10:58.000
But this is what we considered our computational budget to be.

00:10:58.000 --> 00:11:01.000
So we we wanted to render. We thought to make that simple picture.

00:11:01.000 --> 00:11:11.000
It would take about 80 million polygons and these were micropolitons, because we wanted really idi Biddy polycons up essentially in one pixel in size, because it was all that detail there you couldn't get

00:11:11.000 --> 00:11:14.000
by with giant. guns! You need really any bitty, Buck, Huh!

00:11:14.000 --> 00:11:22.000
And you know it just worked out to be like 24 gigaflops to make this image, and i'm not going to go through this whole thing. but it was sort of inconceivable about the time I mean

00:11:22.000 --> 00:11:30.000
i'll talk about this in hardware I mean at the time when Sgi machines they were rendering like 10,000 polygons in real time, you know these like flight simulators.

00:11:30.000 --> 00:11:44.000
I'll show you some pictures of that and so this just you have to, just as a graphics person This has been my world for my whole life that I want like 10 orders of magnitude, and more computing power than I already had to have you know I mean you

00:11:44.000 --> 00:11:48.000
know that just that's what we think it would take to make the simplest possible picture.

00:11:48.000 --> 00:11:50.000
It's not like just some pipe dream right this is based.

00:11:50.000 --> 00:12:03.000
This is an actual calculation back the Emelope calculation that we made at the time, and you know if you look at it today, you know I mean this is some stats I have you is This is pretty. still current.

00:12:03.000 --> 00:12:10.000
But about, you know, over a day of computing per frame in a movie, maybe 2 days, 29 h.

00:12:10.000 --> 00:12:13.000
Okay, a 100 million hours total of cpu time for the movie.

00:12:13.000 --> 00:12:22.000
You can work it out, I guess. my Slides getting blocked off here, but it's about It's about a giga flop, per Pixel, right now, going into a typical movie frame.

00:12:22.000 --> 00:12:30.000
And it just you know it's a kicks incredible monitor computing to do all the stuff you need to do to make that image.

00:12:30.000 --> 00:12:38.000
And and that's just been a driving force in my work through the years, is how to get access to that kind of computing power, and yet still solved.

00:12:38.000 --> 00:12:46.000
This really challenging problem? So in some sense, were our goals at the time.

00:12:46.000 --> 00:12:58.000
And i'm, not going to spend a lot of time talking about renderman. but I did just recently write a paper called the Designer Renderman, which is sort of a fun thing independent of the tournament, but just sort of like how

00:12:58.000 --> 00:13:01.000
I said about designing the system you know just like it's sort of like, you know.

00:13:02.000 --> 00:13:04.000
I really interest in software design might want to read that like.

00:13:04.000 --> 00:13:14.000
Here I am. as a designer. how do I make trade-offs. How do I put together all the pieces in order to make this particular?

00:13:14.000 --> 00:13:25.000
Okay. So anyway, that's just some background. about how computer graphics people think about computation. Or, you know, people mentioned that computer through graphics overlap with computer science in many ways.

00:13:25.000 --> 00:13:30.000
One thing is just how do you build computers that are as fast computers in the world?

00:13:30.000 --> 00:13:33.000
Okay. Another thing we did as part of the renderman system.

00:13:33.000 --> 00:13:41.000
I worked on what I call shading languages. and, by the way to why would I work on Saturday?

00:13:41.000 --> 00:13:45.000
Somebody asked me recently, Have you had any training and programming languages?

00:13:45.000 --> 00:13:54.000
And I said, No, and but that's not quite true Okay, So this is me as a grad student at the University of Wisconsin.

00:13:54.000 --> 00:14:04.000
Notice that this is a wet lab right there's a so that wasn't my idea about that I mean, and I literally taught myself programming in order to do this kind of stuff.

00:14:04.000 --> 00:14:12.000
But anyways oh, sure. How do I get those we need an aircraft?

00:14:12.000 --> 00:14:37.000
Okay, do that, I see? Can also remember the sound. Yeah, see that

00:14:37.000 --> 00:14:42.000
Thank you. Awesome. all right. thanks for interrupting me that's great.

00:14:42.000 --> 00:14:49.000
Okay. So this actually was the first paper I ever wrote at the that report. You can go if you download it from the technical port server at us.

00:14:49.000 --> 00:14:57.000
It's Wisconsin you can actually get it so. but it was. it was called Procedures for parallel array processing a pipeline display terminal.

00:14:57.000 --> 00:15:02.000
And actually it should be a language for parallel ray processing on a pipeline display. turmoil. Okay.

00:15:02.000 --> 00:15:10.000
So the first thing I worked on, so I immediately was interested in computer languages back then, and actually, I mentioned Leonard.

00:15:10.000 --> 00:15:17.000
So my advisor, Leonard, Jr. He was an early Ai researcher.

00:15:17.000 --> 00:15:22.000
And we started working on Ai in the sixtys, and he got very instant on time.

00:15:22.000 --> 00:15:26.000

I was in there in what he called multi-computer architectures.

00:15:26.000 --> 00:15:31.000
For, you know, artificial intelligence, but in particular what he called pyramidal neural networks.

00:15:31.000 --> 00:15:37.000
So here's his book, i'm saying this book is probably published 1980 or so like this.

00:15:37.000 --> 00:15:45.000
I mean, this is not a new idea. Building computers with, you know, pyramidal neural networks, like many layers of neural network, and this was very much more to motivate.

00:15:45.000 --> 00:15:50.000
Of course, by the way, the human brain is architected, and you see, you started seeing.

00:15:50.000 --> 00:15:56.000
So what I was supposed to do was take some graphics hardware that we had, and build a language for doing pure metal.

00:15:56.000 --> 00:16:05.000
Neural networks calculations on it. Now, you know, this thing was incredibly slow by compared to what you know people are using nowadays.

00:16:05.000 --> 00:16:13.000
But the thought that this would be a really efficient way of implementing artificial intelligence, and he wrote a lot of other books which I highly recommend You read.

00:16:13.000 --> 00:16:16.000
I mean. the guy was a genius. way out of his time, like one is.

00:16:16.000 --> 00:16:19.000
His most well-known book, which I think was in the 70 S.

00:16:19.000 --> 00:16:23.000
Was called par recognition, learning, and thought, and he was just.

00:16:23.000 --> 00:16:31.000
Had this view of how to compute on the brain which I think is still would would actually be closer to what we're arriving at.

00:16:31.000 --> 00:16:39.000
Now, which is hybrid and neural networks for symbolic processing, which was his big thing. I just want to mention that so.

00:16:39.000 --> 00:16:44.000
But this idea of shay trees and languages came up in graphics.

00:16:44.000 --> 00:16:52.000
My first invented sort by Rob Cook at lucas film, and what he was interested in doing was simulating different materials.

00:16:52.000 --> 00:16:56.000
So he's very famous for a paper he wrote on Why does copper look different than plastic.

00:16:56.000 --> 00:17:00.000
That's not the official tire But that's basically why do metals look different than plastic?

00:17:00.000 --> 00:17:03.000
And so here we have the same geometry on the right.

00:17:03.000 --> 00:17:10.000
We with different simulated materials, and he would write these little expressions in this language that would sort.

00:17:10.000 --> 00:17:18.000
He could type in some formula as a model of copper versus, for you know, some other material, and this is the one for copper.

00:17:18.000 --> 00:17:26.000
And he was just interested in like you know, having the flexibility to sort of capture all these different kinds of materials so like it.

00:17:26.000 --> 00:17:31.000
Wasn't going to be one material in the world There were a bunch of material part of this renders everything you ever saw, idea.

00:17:31.000 --> 00:17:42.000
So you'd have to and So this actually this little you Could type in these low expressions, and they would get downloaded into rays, and and then you could run them and then get different materials.

00:17:42.000 --> 00:17:46.000
The other thing that was really popular time was procedural texturing.

00:17:46.000 --> 00:17:58.000
So here's an example of a file on paper the next year by a person named Ken Perlin said Inyou, and he was trying to emulate these light, turbulent turbulent kinds of things, and he did that with like a

00:17:58.000 --> 00:18:00.000
fractal noise, texture, and i'll show you an example that.

00:18:00.000 --> 00:18:09.000
And so the motivation the main motivation for these languages was, people wanted to explore computational models of appearance.

00:18:09.000 --> 00:18:12.000
You know it wasn't just physics it wasn't just a formula.

00:18:12.000 --> 00:18:21.000
You could capture and get out of book. You guys have like sort of a computer simulating something are generating a pattern. So here was my language.

00:18:21.000 --> 00:18:33.000
This actually is real code that I wrote i'm making this sort of corroded teapot, and you actually and and you know this, you know, this was in the era of like Unix.

00:18:33.000 --> 00:18:42.000
And yeah and lax, if you know that or you could just roll your own language like John Bentley's little languages, if you remember that sort of thing, you know.

00:18:42.000 --> 00:18:49.000
And there are all these languages on unix like and you know Enron and various other ones like that.

00:18:49.000 --> 00:18:56.000
So, anyway, you know. and so this was this is a typical render Man Shader that you would write and you know what you're doing.

00:18:56.000 --> 00:19:00.000
There is you're adding up a bunch 6 actors with noise, which is bandwidth.

00:19:00.000 --> 00:19:04.000
You have a one number it have fall off to be like a flag.

00:19:04.000 --> 00:19:18.000
The light spectrum, and then that creates this turbulence thing, and then you perturb it surface along the normal by the magnitude of the turbulence at that point which is varying function of space knows that the noise function is indexed by

00:19:18.000 --> 00:19:26.000

position and position is a position on the surface that's being shaded. and now we compute a new position, and then we can new normal.

00:19:26.000 --> 00:19:33.000
Once we deform the surface new normal. Oh, then we can go computer color from that.

00:19:33.000 --> 00:19:46.000
And this is what you would do if you wanted by having this little language. We could add this to the system, and you know there was a bunch of interesting things in this language.

00:19:46.000 --> 00:19:51.000
I'm not going to spend a lot of time on it, but you know we have these sort of like control flow constructs.

00:19:51.000 --> 00:19:58.000
If you like. illuminate and luminous lunate was used by a light source that casts wide into the world.

00:19:58.000 --> 00:20:06.000
So imagine this is like a loop, say, sending out rays of light from that point on light swars, and the luminance was a reverse of that computing.

00:20:06.000 --> 00:20:12.000
The aluminum of incoming light, and computing how much gets reflected in here.

00:20:12.000 --> 00:20:26.000
So these were the sort of 2 fundamental things you had to simulate lights, and you had to stimulate reflection materials, and that's how was expressed in the language in this, you know, as a domain specific language and his design for doing these shaining

00:20:26.000 --> 00:20:40.000
calculations. and you know I just have always been fascinated by designing languages like this for a computing task, because I just think there are very elegant ways of building very extendable systems, and just really fun to think

00:20:40.000 --> 00:20:45.000
through like? what are the abstractions you have, and how do you compose them?

00:20:45.000 --> 00:20:49.000
And things like that. Ok? So anyway, so that's what I sort of did there?

00:20:49.000 --> 00:20:54.000
So we had this so, and the main reason I did that then was I was being.

00:20:54.000 --> 00:20:59.000
I was a little overwhelmed beyond us. So I was a new PHD.

00:20:59.000 --> 00:21:08.000
Graduate, and I was given responsibility for maintaining this renderman software, this race offer and designing the system.

00:21:08.000 --> 00:21:09.000
And remember, you know there were only like 3 of us doing this.

00:21:09.000 --> 00:21:24.000
I mean, was in charge, and I had this huge list of feature requests, like, you know, thousands of features people wanted, and I could either sit down and try to add them one at a time, or I could design and demand this all I knew about computers

00:21:24.000 --> 00:21:27.000
outside of time I could design a language and have them do with it.

00:21:27.000 --> 00:21:31.000
I could design one language and then have them implement all the features, you know.

00:21:31.000 --> 00:21:38.000
And that was very much spirit of computer science. back then you'd build these powerful tools, and it was okay to design new languages back.

00:21:38.000 --> 00:21:42.000
Then. now it's maybe not as okay, but it was okay to do that.

00:21:42.000 --> 00:21:48.000
So it was very much a pragmatic thing it's like, How can I make the system as flexible and extensible as I can?

00:21:48.000 --> 00:21:55.000
Okay, So then, now things start getting instrument so I didn't, quite at the at the same time.

00:21:55.000 --> 00:22:04.000
People started building graphics workstations. So now, so this is correctly, is a PHD student mind.

00:22:04.000 --> 00:22:12.000
But he actually he actually spent. He went off with Jim Clark to found Sgi, and he designed a lot of these early Sgi workstation.

00:22:12.000 --> 00:22:15.000
So he wrote a paper in 1,992 which is around the same time.

00:22:15.000 --> 00:22:23.000
I'm doing this, which is describing all the sgi workstations, and how they sort of fit into different generations.

00:22:23.000 --> 00:22:29.000
So the first one sort of drew wireframes, second ones who did field polygons.

00:22:29.000 --> 00:22:37.000
But the polygons were like constant colors or interpolated colors, and that require a lot more computing, because now you had to write all those pixels into the framework.

00:22:37.000 --> 00:22:41.000
You had to create all those pixels and write them into the frame buffer, and that was like orders of magnitude.

00:22:41.000 --> 00:22:50.000
More computing. Oh, no, that's fundamentally the problem in graphics is like you have something, but you have to do it for every pixel. right?

00:22:50.000 --> 00:22:54.000
So then you have millions of pixels. And now, suddenly one nips machine is a one instruction per second machine.

00:22:54.000 --> 00:23:00.000
So anyway, that did that. And then the third generation was texture.

00:23:00.000 --> 00:23:06.000
Min happy, where you basically place an image on each surface, or made up more than one image on each surface.

00:23:06.000 --> 00:23:15.000
And this requirement even radically more hardware, because you had to now look up the texture and and filter it, and all sorts of other things.

00:23:15.000 --> 00:23:21.000
You had to do the address calculations for the book of the texture, and you had you filter it with alias.

00:23:21.000 --> 00:23:25.000
So these, you know, this would start requiring unbelievable amounts of computing power.

00:23:25.000 --> 00:23:36.000
To do this and one thing that kurt did which I want to spend a couple of minutes on just because I think it's really important Is he defined something called Opengl.

00:23:36.000 --> 00:23:41.000
And after this, and i'm not going to you know again.

00:23:41.000 --> 00:23:51.000
This is by more detail than we can get into, but opengl defines the classic graphics pipeline, which we teach in introgaphic, where you send triangles down the pipeline made up of

00:23:51.000 --> 00:23:57.000
Vertices calculations are done on vertices they've been converted into 2 d triangles.

00:23:57.000 --> 00:24:00.000
Those are rasterized or converted to pixels, or we call those fragments.

00:24:00.000 --> 00:24:07.000
And then those fragments are textured and then they're merged into the frame buffer to be displayed.

00:24:07.000 --> 00:24:14.000
So what was interesting about this was he defined the reason.

00:24:14.000 --> 00:24:28.000
He defined open jails he wanted to sort of emulate a cpu's isa Now, people don't know this, but if but one of the places that I had to do see if I brought school you just want to put it

00:24:28.000 --> 00:24:37.000
pass away. But you know he came up with this really important idea is you could specify an architecture independent of its implementation, which was really his idea.

00:24:37.000 --> 00:24:42.000
And by the way, he coined the term architecture in computer science.

00:24:42.000 --> 00:24:48.000
His book is, if you haven't read this book I highly recommend you read it because it's nowadays.

00:24:48.000 --> 00:24:54.000
Computer architecture is how we build a nips machine Now Then it was like here's what it takes to be an architecture.

00:24:54.000 --> 00:24:58.000
And here's what it takes to be an architect it's a more meta thing.

00:24:58.000 --> 00:25:02.000
Now we're just very focused on how to build one particular thing.

00:25:02.000 --> 00:25:07.000
It's like if you define our research on how to build a building, you know, versus some grand scheme about, you know.

00:25:07.000 --> 00:25:16.000
Maybe some idealistic, and he came up the deal with a specification, and and Kurt followed that idea.

00:25:16.000 --> 00:25:26.000
So he said, i'm gonna have a bunch of state including a frame buffer, and i'm going to specify these Api calls, and they're going to i'm going to tell you how they update this same so you

00:25:26.000 --> 00:25:28.000
have a bunch of state, and you have a bunch of things you can do.

00:25:28.000 --> 00:25:34.000
This State to update it, and that and that's very much like the Cpu.

00:25:34.000 --> 00:25:40.000
If you have a cpu through the cpu man you know like an arm manual, it'll say if I implement the ad instruction.

00:25:40.000 --> 00:25:46.000
There are 3 registers I read from 2 of them, and I calculate the sum, and then I add it.

00:25:46.000 --> 00:25:54.000
I write it another register and i've told you how i've updated state given the current State, and the specification of instruction.

00:25:54.000 --> 00:25:59.000
And so he did that for open Di and and and the key thing about this.

00:25:59.000 --> 00:26:08.000
That means that Sgi and others could start building. You know, different implementations of this architecture and people.

00:26:08.000 --> 00:26:19.000
Just, I think, don't realize sometimes, when these ideas catch on in little suburbs of computer science, what an impact they can have this Essentially, let people start building.

00:26:19.000 --> 00:26:28.000
Many different types of graphics systems, including eventually in 1,999.

00:26:28.000 --> 00:26:32.000
A couple years later Nvidia puts a complete graphics pipeline on a chip.

00:26:32.000 --> 00:26:38.000
So so that's a sort of a milestone so and they can turn.

00:26:38.000 --> 00:26:44.000
They came up with a term gpu when they built this thing, Because, they said, Now we have a single chip as a compute graphics pipeline.

00:26:44.000 --> 00:26:53.000
By the way, that's an isa that can be evolved forward that can get better and better with new generations just like at X 86 can get better and better.

00:26:53.000 --> 00:27:03.000
And and now it's sort of a standalone thing right. Looks like when you had it, you know, Eventually you could put a process around a single check right and to put a prosper on a single chip. you know then, that sort of set things moving right

00:27:03.000 --> 00:27:10.000
So now you had a single chip graphic system, single chip, cpu, gpu, cpu, and so on.

00:27:10.000 --> 00:27:16.000
So. by the way, most people, you know, if you were to ask anybody outside anywhere they would not know.

00:27:16.000 --> 00:27:20.000
This is going up. ask a graphic I mean an architect at this time, you know.

00:27:20.000 --> 00:27:23.000
This is just weirdness going on on the side, you know.

00:27:23.000 --> 00:27:32.000
But it was pretty significant. So so this was happening right when I got that research infrastructure guy.

00:27:32.000 --> 00:27:38.000
But I basically had another grant from Darpa, which was basically I could just.

00:27:38.000 --> 00:27:42.000
And this was for my ground. I recently said we should try to run these real-time chasingly.

00:27:42.000 --> 00:27:45.000
We should link shame on which is run in real time.

00:27:45.000 --> 00:27:50.000
On these chips and and you know in general a lot of technology.

00:27:50.000 --> 00:27:54.000
It's not that hard to forecast the future if you have this one particular thing.

00:27:54.000 --> 00:28:02.000
You have some batch-oriented process going on, forging a head, developing new algorithms and capabilities.

00:28:02.000 --> 00:28:12.000
And then you have this need to go to real time and Then you just sort of There's some leg between the 2 as the computer power catches up.

00:28:12.000 --> 00:28:20.000
And so you can sort of say, you know, we sort of know this is a way to do it, and and mill, and then you implement it.

00:28:20.000 --> 00:28:28.000
Now we were lucky in graphics so that was like a 10 year leg period like that's going on in ai right now, but it's by like a 6 month long period, or something like that right?

00:28:28.000 --> 00:28:31.000
So I mean you can't forecast it and really think it through.

00:28:31.000 --> 00:28:38.000
But we could. We could do that here So that sort of setup.

00:28:38.000 --> 00:28:51.000
Yeah, I'm giving you the service historical thing by the way, the reason I put together this talk from a historical perspective was because I read John Solston Sydney Brennan's and all these Nobel

00:28:51.000 --> 00:28:55.000
lectures, and I also wrote up on Saturian lectures.

00:28:55.000 --> 00:29:08.000
There was really something quite striking about them the nobel lectures are all Here's what I did Here's how I discovered the electron, you know Here's how I figured out program cell death you know Here's

00:29:08.000 --> 00:29:14.000
what I did the Turing lectures are all over the place they're like essays on things you know what I mean.

00:29:14.000 --> 00:29:19.000
So I decided to go back to my roots in some sense, and just sort of tell you what I did. so.

00:29:20.000 --> 00:29:27.000
Anyway. This was a really interesting period of time. Yeah, no the time scales here. These are just long enough time scales.

00:29:27.000 --> 00:29:33.000
They're not like paper projects you know it's not one paper right?

00:29:33.000 --> 00:29:39.000
This is like one or 2 PHD. Thesis 5, 6, 10 years right, and where you keep working on it until you get it to work, you know.

00:29:39.000 --> 00:29:43.000
So anyways, this is sort of how we got to real time chain language.

00:29:43.000 --> 00:29:51.000
So spi has We've been called the past Okay, Then we built a Stanford real-time shading language and kick color.

00:29:51.000 --> 00:29:56.000
Kaufman Bill Mark, or the main 2 on that.

00:29:56.000 --> 00:30:00.000
And then we transferred that technology to Nvidia. Phil.

00:30:00.000 --> 00:30:09.000
Mark went in and did, and then Curb. Akley most part. I don't know mentioned him for that minute, and they came up with this Tg.

00:30:09.000 --> 00:30:19.000
System which became H. Lsl, Mg. and that was sort of the first real time implementation of a Savior language on the Gpu.

00:30:19.000 --> 00:30:24.000
So I think this might be a little bit too much detail for this audience.

00:30:24.000 --> 00:30:31.000

But if he can give me some feedback, maybe but or ask questions. I'm going to go over this sort quickly.

00:30:31.000 --> 00:30:38.000
But there were a couple interesting points here for the architect types in the audience like Margaret. Yeah.

00:30:38.000 --> 00:30:45.000
So the first thing is that the way they first people first thought of doing this was what we call multi-pass algorithms.

00:30:45.000 --> 00:30:51.000
If you wanted to do a complicated image of a bowling pin, you would draw in multiple passes.

00:30:51.000 --> 00:31:01.000
So you draw the bowling pin like 6 times and every time you draw the bowling pen. you would apply like a texture to it or some function to it.

00:31:01.000 --> 00:31:05.000
Okay, so it's like like over is like sort of a pilot decount or something.

00:31:05.000 --> 00:31:14.000
So you go over it, you over what you draw. This is on the base tensioner, and then eventually you add in wisdom, shaving, and so on.

00:31:14.000 --> 00:31:19.000
But you start, you know, wanting multiple countries to keep on it over and over, and accumulating the result.

00:31:19.000 --> 00:31:32.000
And what you think of is opengl is a big parallel computer where you have the frame buffer which access the accumulator, and you send these passes down there, which are like one instruction.

00:31:32.000 --> 00:31:36.000
And then, you know, you have access to the texture T, and the color of the fragment.

00:31:36.000 --> 00:31:39.000
You have some operation that combines and accumulate framework.

00:31:39.000 --> 00:31:45.000
Then you're generating, and you can also copy the frame buffer the texture.

00:31:45.000 --> 00:31:51.000

So it can be used anyway that's the first way people thought of doing this.

00:31:51.000 --> 00:32:02.000
And the second way that people thought of doing it. was what we were advocating was, you took the open gel graphics pipeline. If you just consider programmable stages.

00:32:02.000 --> 00:32:13.000
So he said, i'm going to put a little program in this state, going to sit in between the fragments coming on the raspberry pi or the combined framework, and it's going to have access to textures and

00:32:13.000 --> 00:32:19.000
registered that it can run that's a sort of original word Shader.

00:32:19.000 --> 00:32:34.000
You're just gonna drop in a little program here and you know the the second way turned out to be a lot better than the first way, and and that's because the first way was sort of like operating on vectors like vector like

00:32:34.000 --> 00:32:41.000
and ducking out there. I think it's coming But you know you take a vector and you sum them up or multiply them.

00:32:41.000 --> 00:32:52.000
And the problem with that is you like, Read 2 flows to one operation and write one float, and you just use a lot of bandwidth and very little computer.

00:32:52.000 --> 00:32:57.000
And it turns out these vectors can't be cashed because they're like whole images.

00:32:57.000 --> 00:33:01.000
So they won't fit in so it says computers increasing much faster than the memory bandwidth.

00:33:01.000 --> 00:33:15.000
And so vector our functions are used in our town and at least By the way, we've measured gpus at the time, and you could do about 12 floats in operations for every float.

00:33:15.000 --> 00:33:29.000
You read. I mean this was been years ago. so you just can't do that much memory bandwidth practically, And, by the way, Dpu spend a lot of money on memory bed compared to Rome, so the other way.

00:33:29.000 --> 00:33:33.000
To do. It is Bill Dalley and I came up with this term because we were trying to explain to people.

00:33:33.000 --> 00:33:36.000
We called it arithmetic intensity. You just want to organize your computation.

00:33:36.000 --> 00:33:43.000
So you do as money operations as possible for a number of bytes, and we were assuming you were like streaming these damage.

00:33:43.000 --> 00:33:54.000
You work for me like the program and randomly send a little packet program over it. That program should do a lot of computers that should right now.

00:33:54.000 --> 00:33:58.000
And and so you know that that we called the you know.

00:33:58.000 --> 00:34:05.000
Just change this a little bit. Call this function F, and make F really big and fat, and then you had way of doing it.

00:34:05.000 --> 00:34:09.000
And so it what our paper on the Rtsl and the Cg.

00:34:09.000 --> 00:34:21.000
System was basically trying to do that was trying to not do this multi-pass stuff to break out of that vector thinking and do this more stream oriented thing. Okay, anyway.

00:34:21.000 --> 00:34:30.000
So that i'm just sort of getting i'm sorry just telling you what the thought process was on these things because it that in any of the papers really you know.

00:34:30.000 --> 00:34:38.000
But we were very aware of these things at this time. I was working with Bill Dalley, who want to now, you know, works at it.

00:34:38.000 --> 00:34:42.000
I mean that, Nvidia. We worked this and this together.

00:34:42.000 --> 00:34:46.000
Okay, So there was one more step and then i'll be finished.

00:34:46.000 --> 00:34:54.000
Okay, General: purpose, Gpu So well, okay.

00:34:54.000 --> 00:35:03.000

So this shows you your age, I guess. But when I was in school, you know, I showed you we had these parallel computers all over the place when I was in school.

00:35:03.000 --> 00:35:07.000
It was a hot thing, pyramidal image processing.

00:35:07.000 --> 00:35:13.000
The connection machine was around at that time. you know there's all this, you know, parallel computer.

00:35:13.000 --> 00:35:18.000
I learned all the way back, you know, from the 60 s like Batchers, you know.

00:35:18.000 --> 00:35:24.000
Parallel sort, you know, attending source so I mean it wasn't like parallel computing was, like you know, some new idea.

00:35:24.000 --> 00:35:32.000
It was like goes back to the birth of computing but you know it's sort of one of these ideas that sort of gets lost like a lot of ideas.

00:35:32.000 --> 00:35:41.000
But being old enough, said, Well, why don't instead of sort of piece moving these CD programs together to do various things?

00:35:41.000 --> 00:35:44.000
Why don't we just try to make the gpu into general purpose?

00:35:44.000 --> 00:35:52.000
Parallel computer that a problem computer, And so you know, and this was a a few years where we tried to do this.

00:35:52.000 --> 00:35:57.000
So it was pretty obvious from day one if you draw a triangle and run a shade run all the triangles.

00:35:57.000 --> 00:36:00.000
That's a little bit like the map if you know you know map from functional programming.

00:36:00.000 --> 00:36:07.000
You know, you map a function over a collection, and we could do that by running a shader.

00:36:07.000 --> 00:36:11.000
Okay, Another thing we could do really easily was remove items from the collection.

00:36:11.000 --> 00:36:15.000
So. so I guess the lesson from data parallel programming was.

00:36:15.000 --> 00:36:22.000
There were a small number of well-known primitives, and if you implemented the small number of wildlife parameters, you appeal.

00:36:22.000 --> 00:36:28.000
It already forged to head and implement in many, many algorithms, fluid, flow, molecular dynamics.

00:36:28.000 --> 00:36:30.000
And this was all well trotting territory, you know.

00:36:30.000 --> 00:36:35.000
People running them on praise, connection machines, other parallel computers.

00:36:35.000 --> 00:36:47.000
So if you could implement these small number of algorithms or higher level primitives, then you'd get to the point where all this knowledge that we already had would just sort of become usable.

00:36:47.000 --> 00:36:58.000
Ok. so a tricky one was, was gather So you have a collection of memory addresses like an array of address, and you want to load in all the values at those different arrays.

00:36:58.000 --> 00:37:02.000
That's very common in these programs. So we could sort of Do that with texture lookup.

00:37:02.000 --> 00:37:10.000
If you think of the texture as memory if you said you know access one of the elements of the texture that's like doing a load right?

00:37:10.000 --> 00:37:19.000
So we sort of had that. But I want to say something about that, because this is the key thing that made Gpus useful, and that is okay.

00:37:19.000 --> 00:37:35.000
So if you look at this, so here here was an example of a little shader, and this was the shader that people would benchmark their gpus again. there'd be one texture load and about 5 other instructions

00:37:35.000 --> 00:37:39.000

so it'd be like if you had a normal processor and you every fifth instruction.

00:37:39.000 --> 00:37:46.000
You would do a load from a random memory address. So you have learned in a little penny program does very little stuff there.

00:37:46.000 --> 00:37:50.000
3 5 instructions, but loads from a random memory it doesn't hit the catch.

00:37:50.000 --> 00:37:55.000
Those from random variable. Well, that tends to. I see markets are to squirm a little bit.

00:37:55.000 --> 00:38:00.000
Okay, that makes people nervous because you know that just doesn't work.

00:38:00.000 --> 00:38:10.000
Very well. Okay. So So short in the loop texture reads from randomly doesn't cash full time for a locality that's like the nightmare program from an architect's point of view that's the program

00:38:10.000 --> 00:38:18.000
we had to get to run on a gpu. So so how did they solve that?

00:38:18.000 --> 00:38:20.000
I wrote a bunch of papers on this how to solve this actually.

00:38:20.000 --> 00:38:35.000
So what you did at the time was you had the raspberry pi you would output on these fragments, and they put it in a fifo, and then the blues that send out to the texture memory the address that it wanted

00:38:35.000 --> 00:38:39.000
to access, and that, you know, pathway to memory and back.

00:38:39.000 --> 00:38:44.000
It takes a lot of time. You just keep rendering fragments as it does that.

00:38:44.000 --> 00:38:55.000
And then eventually this this thing comes out of the python and you make this Bible large enough so that has time to do a picture of the fetch. and then the data comes back and you words in it.

00:38:55.000 --> 00:38:59.000
That's how it works. you know you had to put in this big firefoot Here.

00:38:59.000 --> 00:39:06.000
But yeah, it was perfectly easy to do in a gpu and that's all the gpus work.

00:39:06.000 --> 00:39:14.000
So that's interesting. So anyway, that evolved in a gpu multi-threading.

00:39:14.000 --> 00:39:21.000
So the way modern computers work is that they have a bunch of fragments being processed like 30.

00:39:21.000 --> 00:39:25.000
They do the final instruction program. they do the texture load.

00:39:25.000 --> 00:39:37.000
They really stop that threat done for the next one. Meanwhile, that that texture, memory, access has been processed, and then they just keep doing that.

00:39:37.000 --> 00:39:41.000
And this is called multi-threading. there are lots of machines built like this.

00:39:41.000 --> 00:39:50.000
But we just just assume you're going to Miss is the soon you're gonna miss the tent look at it. Who you're doing this just really junk the next time.

00:39:50.000 --> 00:39:55.000
And then finally, after you have enough spreads, run this case at least 48 threats.

00:39:55.000 --> 00:40:09.000
So there's a lot of these then you come back and start funny, and you know the downside of this is that you know each of these threats can't have a lot of state because you have to have 40 sets of registers

00:40:09.000 --> 00:40:13.000
or whatever, in order to have all these threats, which is a real pain.

00:40:13.000 --> 00:40:20.000
Now these things have to happen really small programs very little state but that's how they solve the problem.

00:40:20.000 --> 00:40:26.000
But that's what ended up happening and so then what happened is around this time.

00:40:26.000 --> 00:40:32.000
Gpus became multicorecindy, multi-threaded machines.

00:40:32.000 --> 00:40:39.000
So basically, you know, at the time, and by the way I was at safer. We had one architect that did one of these things.

00:40:39.000 --> 00:40:43.000
Another architect that did another one actually we didn't have a multi-threaded guy.

00:40:43.000 --> 00:40:47.000
Okay, we definitely had a multi-core guy. We definitely had the Simd black, and they they were like in different world.

00:40:47.000 --> 00:41:01.000
Now here's how we built this right totally different world and they went. And I would say, Well, let's say we should do all these things because we need everything we can get okay, and that's what they did And that just shows you how crazy

00:41:01.000 --> 00:41:08.000
graphics people, and just because we needed we needed those cycles.

00:41:08.000 --> 00:41:20.000
We could not. We were not going to give up a factor of 20, or something, if we, if it was just a matter of paralyzing our program more, we wanted the power.

00:41:20.000 --> 00:41:28.000
We were willing to you know. Do extreme stuff. So anyway. Then that sort of evolved in what we think of as the modern.

00:41:28.000 --> 00:41:36.000
There's one more step in that. which was It started out that these shaders there were multiple different kinds of shaders for different parts of the graphics pipeline.

00:41:36.000 --> 00:41:45.000
They came up with this idea of a unified changer, where all of them would execute the same instruction, set and run on the same hardware.

00:41:45.000 --> 00:41:49.000
In that case came out with Nvidia G. Force 8 anyway.

00:41:49.000 --> 00:41:53.000
So this is, you know. So you have 2 levels of parallelism here. One is.

00:41:53.000 --> 00:41:57.000
You have number of cores as a number of student, etc.

00:41:57.000 --> 00:42:05.000
But then you have that times a number of threads before, and that gives you the friends

00:42:05.000 --> 00:42:12.000
So but anyways, you know, then things were pretty much golden, I mean.

00:42:12.000 --> 00:42:22.000
So this this gather I mentioned the gather thing because I think that's why Gpus became so ubiquitous.

00:42:22.000 --> 00:42:32.000
Is, they had that gather. There were lots of computers like Dsps, and a lot of these deep learning things that don't have this random fact for memory in the middle of their inner loop, and they're going to be

00:42:32.000 --> 00:42:46.000
restricted in the programs they can walk. So they thought the scatter thing, This texture mapping sort of put gpus a bit of a sweet spot, and enabled them to run a lot more things than other types of machines So

00:42:46.000 --> 00:42:51.000
anyways, to complete the full, you know Set, you also need a scatter.

00:42:51.000 --> 00:42:57.000
Let them talk about that. You need some way of doing a reduction scan or fold.

00:42:57.000 --> 00:42:59.000
I'm not going to talk with that but it's fairly easy.

00:42:59.000 --> 00:43:13.000
We kept working on this, and others kept working on it and pretty soon. Bingo. Okay, And then 2,004 paper with Pian bug, you know a book for Gpu's Dream to put it on bathley's

00:43:13.000 --> 00:43:20.000
hardware where we basically say, well, we can build a data parallel virtual machine on top of all these graphics primitives.

00:43:20.000 --> 00:43:27.000
Now you can program this thing as a computational scientist or an Ai researcher don't have to know anything about graphics.

00:43:27.000 --> 00:43:39.000

And then, and video hired. i'm dealing with finish this thesis, and we're done. John Nicholas was the hard work team leader.

00:43:39.000 --> 00:43:53.000
So So you know they're they're sort of this fairly obvious progression of ideas here. you know none of them knew, like none of my work has ever involved any new ideas as I like to say, none of them.

00:43:53.000 --> 00:44:00.000
Knew, just taking old ideas mostly, and not getting rid of them, or prematurely, you know. Sometimes they just sit around in the back of your mind for many, many years.

00:44:00.000 --> 00:44:05.000
You keep working on it, You know. I always knew we needed more computing power.

00:44:05.000 --> 00:44:10.000
Just keep working on it. Okay, How much time do I have?

00:44:10.000 --> 00:44:17.000
And how much? Oh, okay. So I can spend a couple more minutes.

00:44:17.000 --> 00:44:21.000
I guess i've been talking about so Ok. no interruptions either. Ok.

00:44:21.000 --> 00:44:24.000
So maybe just 2 implications of this and then i'm done.

00:44:24.000 --> 00:44:28.000
Okay, So and you know then, you've probably seen talks about this.

00:44:28.000 --> 00:44:41.000
But I think it's so i'm a huge fan of domain specific languages, huge fan of them most computer scientists and lost software engineers are not a fan of them i'm somewhat surprised So that's Why, I'm: going to

00:44:41.000 --> 00:44:46.000
pitch them a little bit here. but so here's gonna give me an example.

00:44:46.000 --> 00:44:49.000
So here, I said. I described opengl to you before.

00:44:49.000 --> 00:44:52.000
Let me tell you what it looks like. from reporters point in view.

00:44:52.000 --> 00:45:00.000
You write these little programs, these C programs, and I love to program in open geology, And I love to make pictures with computers and write programs like this.

00:45:00.000 --> 00:45:05.000
So it makes me happy. Just look at that code. Okay, But you know you set up a camera.

00:45:05.000 --> 00:45:13.000
You draw a bunch of triangles and you swap buffers, or complete your image. it's so obvious what this does right.

00:45:13.000 --> 00:45:19.000
It's pretty easy to learn pretty clean syntax and so on.

00:45:19.000 --> 00:45:22.000
But it's really like a language even I mean it looks like just a bunch.

00:45:22.000 --> 00:45:30.000
It looks like a normal Api. but it's really like a language it has like a grammar to it, like you have to do things in certain orders, right?

00:45:30.000 --> 00:45:37.000
I mean you have like. If you want to draw the final, you have to say again 12 vertices, and then n, and then make the flow under our vertex.

00:45:37.000 --> 00:45:44.000
Keep it on 1 one vertices you get to call this have a normal color. I mean, there's there's an order to it makes sense.

00:45:44.000 --> 00:45:48.000
I mean it's a grammar to it and if you don't obey the grammar.

00:45:48.000 --> 00:45:51.000
There's a parser and it and it crashes your machine.

00:45:51.000 --> 00:45:58.000
You're sending it This thing and it says that's not a legal open jail program, and it gives you an exception.

00:45:58.000 --> 00:46:02.000
So it has this language like thing, even though it's embedded in C.

00:46:02.000 --> 00:46:06.000
So I will call that a Dsl embedded in C, as function calls.

00:46:06.000 --> 00:46:20.000
So a lot of libraries are just dsl really because they have some syntax, and they have some grapper, and and they have some semantics synthesis with a grammar and semantics, so that's

00:46:20.000 --> 00:46:23.000
sort of obvious. but people tend to think oh, that's not a Dsl.

00:46:23.000 --> 00:46:30.000
But it is a Dsl: Okay, I mean it has grammar and a smith that's what i'll call lady

00:46:30.000 --> 00:46:33.000
Okay, and why is this Such a good idea it's easy to use.

00:46:33.000 --> 00:46:40.000
It runs on all these different machines, and it runs really fast.

00:46:40.000 --> 00:46:50.000
All sounds good, but the thing about it in graphics is because we were implementing this really high, level, opengl architecture.

00:46:50.000 --> 00:46:55.000
It encouraged incredible amount of innovation. So what was happening?

00:46:55.000 --> 00:47:01.000
I think, around that time in in cpus, as they were sort of getting stuck, running c.

00:47:01.000 --> 00:47:11.000
Programs. they couldn't really figure out a path forward to parallelism, and also if they tried one form of parallelism, then you have to rewrite your C programs in one way.

00:47:11.000 --> 00:47:20.000
If you had tried another form parallelism, you have to rewrite your C program in another way, and and that was really difficult to convince people to do with using opengl.

00:47:20.000 --> 00:47:36.000
There could be different parallel machines underneath the hood, and it was up to the vendor to implement the the system, and once they implemented it, the opengl programs as well. And so during this period of time the gpus just kept

00:47:36.000 --> 00:47:40.000
changing radically, radically. But all your open jail programs just kept running.

00:47:40.000 --> 00:47:52.000

It's still easy to use really performant and so on so I think that would not have happened without opengl Cuda, and i'll say this a little bit. having been part of starting.

00:47:52.000 --> 00:48:04.000
It is a bit of a step backwards. Now we've actually come up with this fairly precisely defined data parallel machine, which you know is hard to use.

00:48:04.000 --> 00:48:10.000
You know, writing coter programs is very difficult for most people. just too much computer science.

00:48:10.000 --> 00:48:21.000
Now you need to know too much architectural knowledge and so in some sense we've regressed a little bit. I mean we we've raised the bar for performance, but we haven't kept the usability in place We made it very

00:48:21.000 --> 00:48:27.000
difficult rights program. So I believe even now, and a lot of my work recently has been building Dsl.

00:48:27.000 --> 00:48:32.000
Serve on top of Cuda that try to make this much more easier to use for people.

00:48:32.000 --> 00:48:41.000
And you know you can see this in Ai with things like pie torch, and so on, where they built ai libraries on top of Kuda that led them doing the on earth.

00:48:41.000 --> 00:48:50.000
So i'm a big fan of that and Then the same thing domain specific architectures by this domain specific thing.

00:48:50.000 --> 00:48:54.000
Maybe that's also a little bit my biology training you know in biology.

00:48:54.000 --> 00:48:58.000
Biologists have an appreciation for the diversity of the world.

00:48:58.000 --> 00:49:01.000
They don't like monocultural world They don't like a world just full of flies.

00:49:01.000 --> 00:49:09.000
Would be a pretty borough boring word, so if we only have one architecture, one kind of computer to a biologist. That would be pretty boring, you know.

00:49:09.000 --> 00:49:15.000
You you imagine we'd have like an ecosystem of kinds of computing devices all evolved for specialized.

00:49:15.000 --> 00:49:23.000
That's what I would call domain specific or evolved or specialized fit, if you like that term fit for the task.

00:49:23.000 --> 00:49:28.000
But anyway, this is a big thing nowadays because of this end of Moore's law, right, which you all know about.

00:49:31.000 --> 00:49:32.000
I'm not going to get into it too much but if you know if you look at I don't know.

00:49:32.000 --> 00:49:41.000
Have they come and spoke here. I don't know but they've given this talk a bunch of times recently, but they think that the end of Mall's law is actually a golden age of computer architecture.

00:49:41.000 --> 00:49:49.000
And it back to biology. analogy. was this thing in biology called the Cambridge Explosion, where you suddenly get all these new life forms.

00:49:49.000 --> 00:49:57.000
That's what I think when when I read this title is we're being forced to actually come experiment with different architectures, different ways of doing things. Okay?

00:49:57.000 --> 00:50:04.000
And so they think domain specific software and hardware. is the way to get more computer performance in an era.

00:50:04.000 --> 00:50:12.000
When Moore's law is forcing computers to sort of not increase exponentially in performance, So you know.

00:50:12.000 --> 00:50:17.000
And this is all obvious, I guess. but you know I did.

00:50:17.000 --> 00:50:24.000
I spend both time with apple these days helping them they're really in hardware softwareical design. But you know this is their latest M.

00:50:24.000 --> 00:50:31.000

One ship. there's men, i'm not going to tell you everything that's in the city like 30 different specialized processors in this chip.

00:50:31.000 --> 00:50:36.000
One thing I will tell you about just the center notice that Gpu in the center.

00:50:36.000 --> 00:50:44.000
It's way bigger than the cpu and Now it knows who's right? Next to the memory system the Gpu and the Cpu is is not a computer.

00:50:44.000 --> 00:50:52.000
It's it's a you know user interface engine basically. So so all the heavy confusion is being done by the Gp.

00:50:52.000 --> 00:50:56.000
And, in fact, I think this will eventually evolve into even more they'll have 2.

00:50:57.000 --> 00:51:01.000
So general things. The Cpu are the cpu but i'll say the core Tp.

00:51:01.000 --> 00:51:05.000
You call that a compute engine, All that data parallel computer.

00:51:05.000 --> 00:51:20.000
So you have a Cp sequential computer signed one single side programs past the puzzle gpu, which is fine, that a parallel program is responsible. Then Also, a bunch of specialized things for when things are kept month even the

00:51:20.000 --> 00:51:31.000
Tpu, like decoders of security, protocols, encryption or speech Recognition, G. P.

00:51:31.000 --> 00:51:38.000
T 3 anyways. So yeah, So that's definitely the trend and everybody knows that.

00:51:38.000 --> 00:51:44.000
But so, anyways, that's just domain-specific things are going to be more important in the future.

00:51:44.000 --> 00:51:57.000
In computing male I'll just end with that I mean specialization will become much more continent computer scientists. I that I talked to about this through many years.

00:51:57.000 --> 00:52:06.000
Don't like this idea they like general purpose they like the fact, there's one way to do things there's one language.

00:52:06.000 --> 00:52:13.000
There's one X one operating system this one you know there's one model of computation, one model of computation Turing machine.

00:52:13.000 --> 00:52:19.000
That's our model of competition so you know that people don't like that.

00:52:19.000 --> 00:52:27.000
It doesn't bother me at all but you know people and we don't want to be more specialized.

00:52:27.000 --> 00:52:32.000
We want to be is as general as as general as possible, not to general.

00:52:32.000 --> 00:52:38.000
And anyway, and then you know i'll just mention 2 other things I said it.

00:52:38.000 --> 00:52:48.000
I think what I learned from my advisors were the choose challenging problems and to use appropriate methodology, or to use appropriate techniques.

00:52:48.000 --> 00:52:57.000
And so, you know, I was very fortunate to just her stumbled into this Lucas film thing where they wanted to make these 4 realistic images, and we had to do it efficiently.

00:52:57.000 --> 00:53:02.000
I mean, it took me 30 years to probably figure out how to do it.

00:53:02.000 --> 00:53:09.000
Now we can run these things in real time. I mean many other people, thousands, tens of thousands of people working on it besides me.

00:53:09.000 --> 00:53:17.000
Obviously, but you know, if you pick a long-term challenging problem, it, it will lead to you know good research. I believe so.

00:53:17.000 --> 00:53:23.000
I hope you keep supporting that The world and then it does take a certain amount of taste.

00:53:23.000 --> 00:53:31.000
I don't know how I got it I maybe I don't really think it get plotted by conventional computer science curriculum.

00:53:31.000 --> 00:53:34.000

But you know there were 2 things I latched onto very early.

00:53:34.000 --> 00:53:41.000
I shall do that first type. Report was language while called language oriented Api's things that have a well-defined semantics.

00:53:41.000 --> 00:53:49.000
For example, a lot of people in computer science says, just come up with these ad hoc implemented like they did sort of bad at being precise about their abstractions.

00:53:49.000 --> 00:53:55.000
It just bothers me a lot of I like these language ideas like, Hey, category theory, abstract algebra.

00:53:55.000 --> 00:53:59.000
You want to get me excited. Talk about stuff like that, and then make it a language.

00:53:59.000 --> 00:54:02.000
And then, you know, parallel architectures I mean the world is kept parallel.

00:54:02.000 --> 00:54:06.000
I knew that I knew the brain was parallel I was telling you about it.

00:54:06.000 --> 00:54:11.000
It wasn't like sequential. I mean maybe I would discuss consciousness with my friends, or whatever.

00:54:11.000 --> 00:54:23.000
But you know we knew the brain was problem. Okay, and I just knew all along there would be many laser into an El Arctic, and I just knew that's the way the world worked and so a lot of people didn't want to deal

00:54:23.000 --> 00:54:27.000
with them. That was a cool faction. So you know, you got a latch onto these really correct abstractions.

00:54:27.000 --> 00:54:33.000
They might not always be trendy but so with that i'll end.

00:54:33.000 --> 00:54:43.000
We'd love to take questions thank you very much so you someday collaborators.

00:54:43.000 --> 00:54:48.000
I'm sure there's some provocative thoughts in there. So hopefully, all right.

00:54:48.000 --> 00:54:59.000
So people who are online feel free to type your questions into the chat, and I will speak them out for you. and people who are in the room feel free to use this microphone as Margaret is about to demonstrate Thank you So

00:54:59.000 --> 00:55:03.000
much. It was wonderful to hear the whole history, and the whole retrospective.

00:55:03.000 --> 00:55:16.000
That was great. So my question is both as an architect and also as the leader of the size director at here at Nsf.

00:55:16.000 --> 00:55:21.000
You know you've succeeded by mixing different topic areas to do what you needed to do right.

00:55:21.000 --> 00:55:33.000
You didn't say i'm a programming languages person, or i'm an architect or i'm a graphics person, you said I want to do photorealistic rendering Yeah, what do I need and So i'm

00:55:33.000 --> 00:55:39.000
curious. What should we do at Nsf to cultivate that style of research?

00:55:39.000 --> 00:55:44.000
So what are the programs we should be offering and out in the rest of the world?

00:55:44.000 --> 00:55:51.000
What should we do to change? How conference communities work in order to cultivate that as well?

00:55:51.000 --> 00:55:54.000
I love that question, and and i'm glad you pulled that out of my dog.

00:55:54.000 --> 00:55:59.000
I actually think I am a fairly problem driven person in general, even though I started out being in science.

00:55:59.000 --> 00:56:04.000
Even then I think the scientific problems I will. When there were problems, You know they were mysteries.

00:56:04.000 --> 00:56:10.000
They were things we wanted to figure out, and we would figure it out, you know, and that started with me at least in physics.

00:56:10.000 --> 00:56:13.000
I mean, I remember, I think modern physics like you know.

00:56:13.000 --> 00:56:23.000
What is there an atom and what's an atom and you know, like Neil's bore and people like that? i'd say weren't like my Hero? So they were sort of they were faced with the reality of the

00:56:23.000 --> 00:56:27.000
world, and they're trying to explain it it was very I consider that someone problem.

00:56:27.000 --> 00:56:31.000
It just didn't know some math and say i'm going to apply my math at general relatively.

00:56:31.000 --> 00:56:34.000
They said, what's the nature of space and time so I'm.

00:56:34.000 --> 00:56:40.000
A fan of problem-driven research. I think and I think you know part of it's practical.

00:56:40.000 --> 00:56:52.000
I'm just a practical person i've started companies as Well, you know, I'm just very driven by making an impact in that process, so that might just be a personality thing I don't know not everybody's like that I know lots of

00:56:52.000 --> 00:57:04.000
people that aren't like that. and I feel very humble when I meet them, because I think they do very pure research, and i'm very practical, you know I do think there's a merit to it, because it keeps you honest and

00:57:04.000 --> 00:57:07.000
keeps you moving forward, you know. I mean you know you can.

00:57:07.000 --> 00:57:12.000
If you work on too specialize of a thing, you can sort of get stuck in your own little world, right?

00:57:12.000 --> 00:57:22.000
Whereas if you're trying to solve a problem you're either solving it, or you, I mean the rubber meets the road somewhere, you're either making progress or you are like if we didn't like continue to make progress on

00:57:22.000 --> 00:57:26.000
making movies, you know, we would have petered out, and we were in mortal fear.

00:57:26.000 --> 00:57:33.000
We would peter out at any given time, because if you know the history of Pixar, I mean, see if Jobs spent almost all his money on it.

00:57:33.000 --> 00:57:37.000
And it took. It took a long time like 20 years to make that movie.

00:57:37.000 --> 00:57:41.000
And so, but we were making progress. We were pretty disciplined about it.

00:57:41.000 --> 00:57:44.000
We evaluated our progress. So you, you you know.

00:57:44.000 --> 00:57:50.000
And yeah, we would do whatever it would take so yeah so I don't.

00:57:50.000 --> 00:57:53.000
I don't know I mean I I do believe that's

00:57:53.000 --> 00:58:07.000
This is a tough question to answer So i'm going on but I do believe you know fundamental things you know are really important to invest in, because if I think of my career, even though maybe I am somewhat problem-driven I've been

00:58:07.000 --> 00:58:18.000
able to cherry pick great ideas from people that have been working at the core, and maybe maybe we're so focused on the core that they didn't go imply them to various areas that they could've internally in

00:58:18.000 --> 00:58:23.000
architecture, and in Pl. I used to give talks all the time like.

00:58:23.000 --> 00:58:26.000
In fact, I gave a talk. Why are Gpu so fast?

00:58:26.000 --> 00:58:40.000
I think i've given that talk 100 times like 20 years ago, and I was trying to get pl people and architecture. people interested in it, and you know, I think they were just so busy working on their area that this was just too.

00:58:40.000 --> 00:58:48.000
You know Maybe that's interesting, Maybe it's not interesting I don't know, you know I I can understand why they might not be part of it.

00:58:48.000 --> 00:58:53.000
So I don't exactly know I mean I just think you need a balance, maybe, is what i'll conclude with.

00:58:53.000 --> 00:59:00.000
I mean you need to support the core areas. but you want to be You want to solve real problems of high impact.

00:59:00.000 --> 00:59:03.000
I mean if you're not solving any problems of high impact that's the other thing.

00:59:03.000 --> 00:59:08.000
If you aren't solving any problems of high impact then something's wrong, you know.

00:59:08.000 --> 00:59:13.000
If you're investing in all the core stuff and you're solving problems of high impact, as spin-offs right and left.

00:59:13.000 --> 00:59:19.000
Then there's nothing to worry about but If you're never solving any problems of any importance.

00:59:19.000 --> 00:59:25.000
I think something's wrong, and so you maybe just have to keep recording that.

00:59:25.000 --> 00:59:29.000
And yeah, so same, thank you for the question I don't know if I answer it very well.

00:59:29.000 --> 00:59:34.000
But I appreciate it a lot. Yeah, it's something to think about. thanks for that.

00:59:34.000 --> 00:59:39.000
So I have a question from the chat. This is from Aruna Kalaru, who says, Thank you, Dr.

00:59:39.000 --> 00:59:46.000
Hanrahan This is her nukelaro i'm a program director in biology division.

00:59:46.000 --> 00:59:52.000
I am curious to know how your background in biology influences your thinking and career track.

00:59:52.000 --> 00:59:58.000
Okay, that's a good question. so yeah I was definitely a hardcore biologist.

00:59:58.000 --> 01:00:11.000

Well, I was not a very successful biologist so, but my background, my My course work was largely in biology and physics, and you know I worked in a wet lab.

01:00:11.000 --> 01:00:23.000
I tried to make electrodes and stick them. in your hands. and I went to the slaughterhouse and picked nematodes out of pigs. and that sense I mean I yeah, I was I was a biologist. Okay, So you know so I don't

01:00:23.000 --> 01:00:29.000
I think the biologists Well, one thing is, it was you when I was in ballology.

01:00:29.000 --> 01:00:32.000
Theoretical biology was not a big thing. In fact, they hated their neural nets.

01:00:32.000 --> 01:00:36.000
Actually I had this one advisor who was totally Mr.

01:00:36.000 --> 01:00:41.000
Neural net, and my other advisor was studying the neurobiology of Inverbrates.

01:00:41.000 --> 01:00:53.000
He was completely adioned on that and the biologists really didn't like to abstract things like that they didn't like a simple model like that, and a website. so I was in the only that time we're talking

01:00:53.000 --> 01:01:04.000
about like in the 80 S. Now, I mean, I think the only theoretical biologist that was respected by about just was Francis Crick.

01:01:04.000 --> 01:01:09.000
And you know I admired him. He came to our lab many times.

01:01:09.000 --> 01:01:17.000
So it was this gulf between the practical biology which was very grounded in the real world, like what do we actually know the experimental side?

01:01:17.000 --> 01:01:25.000
And this theoretical side. and so I was that I think that tension is really interesting, and I think that's changing in biology now.

01:01:25.000 --> 01:01:35.000
But I think that really influenced my career was that tension between the reality of the real world and the biological world and sort of this abstract world.

01:01:35.000 --> 01:01:38.000
This theoretical physics world and I think that you know that.

01:01:38.000 --> 01:01:41.000
That's how it influenced me. but I love biology.

01:01:41.000 --> 01:01:47.000
I still follow it pretty closely. I want to do more biology actually, it's fascinating.

01:01:47.000 --> 01:01:51.000
Yeah, I like how you said it before in terms of the domain.

01:01:51.000 --> 01:01:57.000
Specific languages is kind of a biological perspective right to say, why should there be one organism that doesn't make any sense?

01:01:57.000 --> 01:02:08.000
There should be organisms that are adapted to their niches. And that's what a domain specific language is. It's adapting the notion of computation to a particular problem that you have at hand Yeah, And you know if you

01:02:08.000 --> 01:02:10.000
make tools. I do a lot of woodworking and stuff like that.

01:02:10.000 --> 01:02:13.000
You have all these different kinds of saws and chisels and stuff.

01:02:13.000 --> 01:02:16.000
Yeah, I mean, you want general saws that was almost through anything.

01:02:16.000 --> 01:02:23.000
But you also want these really specific songs that let you be really efficient for certain tasks, and it doesn't bother me at all.

01:02:23.000 --> 01:02:30.000
It just seems like that's. the way the world is and biology is a good example. So i've always that's one that I never have liked about computer science.

01:02:30.000 --> 01:02:38.000
They just want these really general things and I just don't need it. But there's there's attention there the right place.

01:02:38.000 --> 01:02:43.000

So even you you don't just write a program you write a language which is a more general construct.

01:02:43.000 --> 01:02:55.000
And so you're you're trying to negotiate this tension between being general and being specific. and you think in some cases this domain specific language is at a sweet spot where it's yeah, you're talking some generality

01:02:55.000 --> 01:02:58.000
but also specificity to the problem and that's true in Biology, too, right?

01:02:58.000 --> 01:03:01.000
There are like like i'm reading this book about the crab cycle.

01:03:01.000 --> 01:03:04.000
Right now. What is Nick Lane or at Nick Lane?

01:03:04.000 --> 01:03:15.000
Great. i'll recommend his boots, a final question transformer But yeah, there's like the crab cycles or or the basic eukaryotic cell architecture or bacteria.

01:03:15.000 --> 01:03:21.000
Pro ki. So I mean, those things are universal and and general, fairly general.

01:03:21.000 --> 01:03:27.000
You know the particular organisms are very different so it's not like you don't have some generality.

01:03:27.000 --> 01:03:32.000
You reuse mechanisms, and you want to learn how you know to reuse them.

01:03:32.000 --> 01:03:49.000
So yeah. So in computer graphics, Now, you talked about a quest for photorealism. one of the ways that we're getting codorealism now is to throw away the graphics pipeline and do this deep learning stuff.

01:03:49.000 --> 01:03:54.000
We have at least one question for Galen. Boden asks Hi thanks for the presentation.

01:03:54.000 --> 01:03:57.000
Any thoughts on opportunities and concerns with emerging deep fake technology.

01:03:57.000 --> 01:04:06.000

So now that people are able to make these photo realistic images just based on data, what is, what do you think about that?

01:04:06.000 --> 01:04:12.000
I mean is something that yeah, that's that's probably the question when I've been traveling around a little bit.

01:04:12.000 --> 01:04:14.000
That's the topic of the day obviously right dolly and Gpt.

01:04:14.000 --> 01:04:18.000
Extract, spent the whole morning playing around with Gpd.

01:04:18.000 --> 01:04:35.000
3. So yeah, I mean well, I think there's 2 questions there. first of all, there's you know, you can make deep fakes with conventional graphics technology, just as easily as with this other technology.

01:04:35.000 --> 01:04:39.000
I mean well, I mean, you know there's there's been a lot of work on defects that preceded Dolly.

01:04:39.000 --> 01:04:48.000
You know what I mean, and that was that's very disturbing work, that deep fake work, and there was a lot of going on, Stafford, and upset me enormously.

01:04:48.000 --> 01:04:57.000
I mean so I think part of that is the fake part of it like, I think if there's an intentionality to it to fake somebody out to mislead somebody.

01:04:57.000 --> 01:05:04.000
And so that that was going on and has always been going on in graphics, and some of it's good like a movie, you know.

01:05:04.000 --> 01:05:14.000
Toy story is fake in some sense but it's good. I think we're not come by other ones we get Obama to say something he would never say, and people would say that's bad.

01:05:14.000 --> 01:05:18.000
So you know, the technology can be used to go back if you make it easy to do bad stuff.

01:05:18.000 --> 01:05:24.000
People will do it. Okay, So that's the first part the Dolly and the Gpt. 3.

01:05:24.000 --> 01:05:32.000
Yeah, I think that's making all of us think about how much knowledge do we need in these systems?

01:05:32.000 --> 01:05:38.000
I mean stable diffusion, and you know transformers and stuff have almost no knowledge.

01:05:38.000 --> 01:05:42.000
And yet they're still able to emulate things that are you know.

01:05:42.000 --> 01:05:56.000
Pretty amazing, or even in chess, you know, with I mean, you know, Alpha 0, you know, has caused the revolution in chess, and it plays unlike any human player has ever played before.

01:05:56.000 --> 01:06:03.000
And it looks crazy. but it's you know it's also generating a sort of a renaissance of chess, so I don't know how this is going to play out.

01:06:03.000 --> 01:06:08.000
But it's you know, and I don't know How much how much knowledge do you need to design a computer system?

01:06:08.000 --> 01:06:12.000
I've been thinking about that because i'm giving this talk like you know, could G. P.

01:06:12.000 --> 01:06:19.000
T 3 just design renderman good question I don't really know I mean, you know.

01:06:19.000 --> 01:06:32.000
Do you really know it's all just random choices we all make, and there's no real reason for anything. So I don't know the answer to my first random man instead of render man exactly But Yeah, it's something we

01:06:32.000 --> 01:06:37.000
all need to think about hope you're funding some work on and ethics of all that.

01:06:37.000 --> 01:06:42.000
And how to think about it. Yeah, thanks thanks for the guidance there.

01:06:42.000 --> 01:06:49.000
So we don't have any other questions online anybody else in the room. have something you want to bring up coming up later this afternoon.

01:06:49.000 --> 01:06:53.000

We have an office hour for Nsf folks to to get a chance to talk to.

01:06:53.000 --> 01:06:56.000
Pat, is that going to be mixed online and in person as well? Yeah.

01:06:56.000 --> 01:07:02.000
Okay, so people can call on for that. Hopefully, you have. Oh, you want to ask, Ok.

01:07:02.000 --> 01:07:06.000
And Dilma will have our last question. Yes, Ok, I think already touched on this.

01:07:06.000 --> 01:07:17.000
But it's about how to educate the people who've been doing maybe the domain specific languages for just sciences and order fields like this intersection. Why worry?

01:07:17.000 --> 01:07:28.000
How going to get there because it seems that the computer science education the traditional one that I know always schools very rare like you said you didn't take Cs courses.

01:07:28.000 --> 01:07:33.000
Many people now graduated with the PHD. that they only took Cs courses.

01:07:33.000 --> 01:07:46.000
Maybe math. and if we're going to keep evolving the world, I think we have to get to someone who will think about the problems that you're thinking now in a domain that is really the domain. So so, if you think about how to better

01:07:46.000 --> 01:07:51.000
educate the next generation of thinkers. Well, I do think in in the languages.

01:07:51.000 --> 01:07:55.000
Example. That's a good question. I think it was a couple of things.

01:07:55.000 --> 01:08:00.000
First of all, we should make it easier to to build demand specific languages and framework.

01:08:00.000 --> 01:08:06.000
So there's a bunch of research in pl going on right Now, that is very relevant to that core.

01:08:06.000 --> 01:08:16.000
I'd say basic research. I mean some of it's really crazy like dependent type theory and independently type programming languages which are really great for building dsl's and you know it's very much at the

01:08:16.000 --> 01:08:20.000
forefront of pl research. So but I mean the pl community.

01:08:20.000 --> 01:08:23.000
I think, completely loves the idea of domestic languages.

01:08:23.000 --> 01:08:28.000
As you get no resistance from talking to them. they love, and they love the embedded Dsl.

01:08:28.000 --> 01:08:32.000
Idea, because, you know, I think I think programming language designers.

01:08:32.000 --> 01:08:47.000
Actually, when I talked to the C sharp team many years ago, and what they thought was the most important thing about our language was to enable building really sort of a libraries, and and they built things, if you remember link this was their query.

01:08:47.000 --> 01:08:49.000
and, like sequel, embedded in C sharp

01:08:49.000 --> 01:08:55.000
So they they were totally into it, and they thought that their job was to be able to build more sophisticated library.

01:08:55.000 --> 01:08:58.000
So I think of domain. specific languages is mostly like next generation.

01:08:58.000 --> 01:09:06.000
Libraries like, How do we build really more powerful libraries, you know, have built in compilers and reasoning about themselves.

01:09:06.000 --> 01:09:09.000
So. So yeah, So they they love it. But there needs to be more research on that.

01:09:09.000 --> 01:09:16.000
Pl. is not as popular as it used to be as a career of investment highly encouraged you to keep investing in it.

01:09:16.000 --> 01:09:21.000
You know it's also very much affects how we think of computing.

01:09:21.000 --> 01:09:25.000
I mean, I think things like Chat Gdp. and how you program in that.

01:09:25.000 --> 01:09:31.000
How you program in some of these advanced languages which have all these auto suggest modes like Hank.

01:09:31.000 --> 01:09:40.000
I've been used egged a lot it's pretty different. and you know just just how we think of programming.

01:09:40.000 --> 01:09:45.000
Just you know how we understand computation. just that's all wrapped up in these topics, you know.

01:09:45.000 --> 01:09:49.000
So I think that's a about the domain specific thing I think the thing to do.

01:09:49.000 --> 01:09:53.000
There would be to invest in joint projects where you have people.

01:09:53.000 --> 01:09:59.000
You need a lot of domain expertise when i've had to design Dsl's.

01:09:59.000 --> 01:10:12.000
I had a doe project where we're working with people doing simulation, and it was really hard to build the right, Dsl: you know, because it had you. It had you correspond to how they thought about their their problem area like what is the

01:10:13.000 --> 01:10:19.000
right level of abstraction. What are the right primitives, you know, and so on.

01:10:19.000 --> 01:10:26.000
And it was a long term process to sort of like think of a way of It's a way of thinking about your field, you know.

01:10:26.000 --> 01:10:27.000
It's like abstraction? like? What are the abstractions.

01:10:27.000 --> 01:10:33.000
What are your primitives? How do you? so? I think it would improve a lot of fields if there was projects like build a Dsl.

01:10:33.000 --> 01:10:47.000
For expressing the computation in your field that would help them formalize how they think about their own field. and at the same time it would inform the computer scientists about what they should be building.

01:10:47.000 --> 01:10:50.000
And they could help them, and there we come up with something useful.

01:10:50.000 --> 01:11:06.000
So you probably need both sides of that. thank you yeah all right thanks so much for the talk.