Peek into the Intel® Architecture Code-Named Larrabee with Tom Forsyth

Tom Forsyth is Intel's software and hardware architect for the Intel® architecture code-named Larrabee. Before arriving at Intel Tom worked on various games for Microprose, Mucky Foot, and SEGA, and also worked at 3DLABS writing Microsoft DirectX® drivers. More recently, he was one of the principals at RAD Game Tools in Seattle, Washington. Tom is known for his extensive work with DirectX and various gaming platforms, as well as for his work over the past four years on Larrabee. In January 2009, Tom took some time to answer a few questions and reveal more about this important project.

Q: *Some have theorized that working on the Larrabee project must be "like being at the absolute center of the gaming universe." Is that how you see it? Has it been exciting to be part of such a large, focused team? And do you interact with other experts both inside Intel and in other companies—can you talk about that a little?*

I'd say it's at the leading edge of the gaming graphics universe. We always have to bear in mind that while graphics are important, they're not what a game is fundamentally about. Games are created and played because they're fun, and graphics are only a part of that. But it's the part that gets the most attention and the most research, and when you include all the art and animation staff, it's where the majority of the effort and development costs go.

One of the most fascinating things I've faced being part of the Larrabee team has been reconciling the needs of hardware design, software drivers, and the game developers who use the whole rendering stack. When considering how to implement a graphics algorithm, we've had an almost unique chance to decide whether a feature is built into the hardware of the chip, handled entirely by the software, or whether it's a hybrid of the two—creating a modified instruction to help accelerate a more general path.

We've worked very closely with a bunch of very smart people. In addition to the public ties we have with people, such as Professor Pat Hanrahan at Stanford and Tim Sweeney at Epic Games, lots of developers in games, industry, and academia have helped us shape the direction of the hardware and software. It's been really fun working with them, seeing their reaction to early designs, iterating on their ideas and wishes, and finally making a lump of smart rock that contains as many of those ideas as we could cram in. And we've barely started discussing the details in public—once the covers come off, I'm looking forward to seeing people do some awesome stuff with Larrabee.

Q: *Intel confirmed that Larrabee is not a pure special-purpose processor or a pure graphics processor, but a product that combines many functions. We've seen that Larrabee will be fully compatible with Microsoft's DirectX engine. Can you describe in general terms your relationship with Microsoft and what its presence has meant?*

I've worked with the DirectX team for many years, starting as a games programmer and then as part of their (large) graphics advisory group, and now I'm on the other side of the fence making hardware and drivers. Microsoft has been enormously important in shaping the industry and ensuring that a broad range of hardware is easily accessible to as many programmers as possible.

That's a tough juggling act—every bit of graphics hardware is different, and each has its strengths and weaknesses. The DirectX team has managed to hide the rather unimportant differences, while letting the really big differentiating factors show through for developers to use.

Although Larrabee can do some really cool stuff that no other graphics card can do, we're acutely aware that the majority of game programmers don't have the time or budget to fully exploit those features unless there is broader, cross-platform support. Full, compliant, and efficient DirectX support is enormously important to the project.

At the same time, we see Larrabee as a powerful tool for prototyping future functionality that we hope other hardware vendors will pick up, which can then be incorporated into DirectX. We're not the first to do this, of course: multi-texturing, hardware transform and lighting, shaders, and then unified shaders were all first available outside of DirectX, either in other APIs or on different platforms, but their adoption by DirectX made them available to the majority of games programmers.

Q: *There are reports that Larrabee is capable of scaling to several thousand cores. When will the actual number of cores supported by Larrabee be revealed? Even without pinning that number down, can you explain why that many cores would interest game developers?*

In the SIGGRAPH 2008 paper we show simulations up to 64 cores, so we're certainly looking at some pretty big numbers, but I think anyone quoting "thousands" is counting something differently in relation to what most people mean by a "core." The large number of cores is an admission that we're hitting some physical barriers. Intel has been making single cores faster for many years by increasing both the clock speed and the amount of work done in each clock. That process certainly hasn't stopped, but it's getting harder.

Larrabee gets its performance by going the other way—making each core simpler, smaller, and more efficient, and then sticking down lots of them. But not too small and simple—you have to strike the right balance between simpler hardware and ease of programming because it's no use having great theoretical peak speed if programmers can't get at it. So that's the tightrope we've been walking with Larrabee. The feedback from developers so far is that we've made good choices.

Q: *Given the huge amount of development that's taken place or is under way with OpenGL* and DirectX, do you see both standards supported going forward, along with a new model? Are open-3D format and solid parallel programming tools enough to drive convergence among graphics developers to build tools and applications that other industries can repurpose?*

On a basic level, there are two extremes of graphics programmers. One extreme wants to make the best graphics possible, and they are happy to pick a single platform and API in pursuit of that goal. These programmers will aggressively use all the coolest new features and jump through any necessary hoops to get the very best graphics they can. They push the limits of graphics on the high-end PC cards and on the single-platform console games. If a graphics card exposes a feature, even through a custom one-off API, they will use it. We love these people, and they will make Larrabee do astonishing, outrageous things, but sadly there aren't many of them.

The other extreme wants to make the best possible game for the greatest number of people. Their game must ship on as many different graphics cards and platforms as possible. These programmers need to pick a set of features that they can support on that wide range of platforms, and to do that they may need to ignore some of the unique features of each platform. To me this

skill is no less amazing—to be able to target platforms as diverse as 20 different PC cards, the Microsoft Xbox 360*, the Sony PlayStation* 3, and the Nintendo Wii*, each with its own little quirks, and to achieve such consistency is truly impressive. Though I guess you have to have tried it and done it to realize just how tricky it is. These people absolutely need every bit of help to make their life simpler, and the OpenGL and DirectX standards are crucial for this. If a feature is not exposed through those standards, they honestly don't have time to use it. Larrabee must have the very best OpenGL and DirectX support possible, because that is what the majority of titles use.

Then there's the middle ground: programmers who mainly use the existing APIs, but have a bit of freedom to play with a few novel features—here and there. Maybe they have a neat lens-flare effect on one card and not another. Maybe high dynamic range is implemented a different way on one card. Or maybe particles are rendered slightly better on one platform than on another. Included in this category is middleware (both public and internal to studios)—when you're making a graphics engine that will be used on multiple titles, adding a few bits of special code targeting specific cards is a reasonable investment of time and effort. So these people need the standard APIs, but they can also explore a bit around the edges and play with some of the new features as long as these features aren't too crazy.

We're supporting all three types of programmers: those who need rock-solid Direct3D and OpenGL support, extended features for those that want to dip their toes in the future, and "bare metal" programming in C++ and even assembly language for those that want to run headlong into the new-but-old world of software rendering.

Q: *As stated at SIGGRAPH 2008, the customizable software graphics rendering pipeline for this architecture uses binning in order to reduce required memory bandwidth, minimize lock contention, and increase opportunities for parallelism relative to standard GPUs. Can you give us a brief introduction to the new types of graphics algorithms that will be possible with this new software rendering pipeline?*

This is a huge area of very exciting research. A lot of people are talking about some longer-term goals, such as Reyes, ray tracing, splatting, and volumetric rendering, which all require new content and new pipelines. But there's also a bunch of exciting near-term features that extend the current rasterization pipeline:

- **RENDER-TARGET READ.** This allows the developer to write a pixel shader that can read as well as write the current target. All sorts of custom-blend operators are possible, not just the small range of fixed-function ones we have today. This sounds like a simple feature, but it's surprisingly difficult to implement in a conventional architecture, and it has some fascinating knock-on effects on shader writing that we've only scratched the surface of so far.

- **DEMAND-PAGE TEXTURING.** We have a full virtual-memory system in Larrabee, which includes every bit of hardware including the texture units. That means that you can read from a texture when not all of it actually exists in memory. Other parts (that you're not currently using) might currently exist on the host system or even on the hard drive. Or they might not exist in a true "texture" format at all—maybe it's a JPEG, a multi-layered Photoshop* file, a Perlin-noise-style procedural description, a PowerPoint slide*, or a text file.

This feature is great not only because it compresses far better than normal texture data, but also because it's easy for the game to change on-the-fly. One interesting example is newspaper headlines that change according to the player's actions in the past. While technically possible now, they're somewhat complex to integrate into an existing game engine.

- **ORDER-INDEPENDENT TRANSLUCENCY (OIT).** Translucent objects are annoyingly tricky to get right in game engines because unlike normal opaque surfaces, the game has to do the depth sorting for the graphics card. Often you can't have many of them, they don't get shadowed or lit properly, or there are restrictions on the type of surfaces you can put them onto. So they are used sparingly in current pipelines, and they're generally treated as a bunch of special cases.

OIT makes translucency a first-class citizen—you render it just like any other sort of surface, and OIT takes care of the sorting for you. Developers have been asking for this for years, but it's very difficult for traditional hardware to implement. Because of the flexibility of Larrabee's all-software pipeline, we can enable this without any extra hardware.

Q: *There was some confusion earlier about whether Larrabee would be rendering graphics using rasterization or ray tracing. You announced on your blog last April that Larrabee was absolutely committed to supporting the conventional rasterization pipeline. Yet ray-tracing would still be enabled for what you described as "wacky tech" projects. What did you mean by that? Is ray tracing still too far out of the mainstream to be considered for cost-justified projects?*

Ray tracing and rasterization are two very different rendering methods, each with its own strengths and weaknesses. Games programmers have been using rasterization almost exclusively for well over a decade, and the art techniques and pipelines are tuned to rasterization's strengths and away from its weaknesses. This also shapes what sort of games people make—where they set them and what they allow you to do. If rasterization doesn't do a certain thing well, people won't tend to make games like that.

Ray tracing requires some significantly different styles of artwork and content to make it shine, and it will allow and encourage different styles of game. And indeed I expect some of those to be very wacky (in a good way!). But that's going to take awhile, and there needs to be broad hardware support before many teams can experiment with these new types of content.

I don't believe ray tracing is inherently "better" than rasterization—it's just different. But one of the real joys of Larrabee is being able to have this discussion at all! We finally have a bit of hardware that can run both algorithms on an even playing field. We soon will have the smartest people in the world writing renderers using their own techniques, and we'll be able to do exact apples-to-apples comparisons and find out.

And better yet, you won't actually have to declare a "winner" at all—you'll be able to have both. Hybrid schemes are already being played with and discussed: Use rasterization for part of your scene and ray tracing for another. It's peanut butter *and* chocolate.

Of course there's nothing new under the sun. Michael Abrash likes to remind me that the original *Quake** engine has two fairly different rendering schemes in it. One is the span-buffer method of occlusion, which is used for the level geometry, and then over the top of that is a more "traditional" Z-buffer scheme, which renders the characters. Each method is more efficient than the other at those particular items of work. It's nice to have that sort of flexibility back in rendering algorithms and to give that power back to developers to see what they do with it.

Q: *You've been right in the middle between the hardware and software engineers bringing Larrabee to market. What has been the biggest challenge so far?*

The biggest challenge is keeping people aware of that balance between hardware and software. People tend to be polarized; they're either hardware or software, and they speak different languages. We always need to keep that dialog going between the two camps. That's my main role through the design phase. Software design can happen much later, so you don't have to decide everything up front. Some things can change after the chip is designed. The amazing thing is that people still ask about changing stuff in Larrabee. Are you kidding me? The architecture has been locked down for a year.

Q: *What is Larrabee's biggest benefit to game developers?*

The primary focus used to be making more realistic graphics. Now as we get there, we see that realism is somewhat overrated. Films don't have real lighting—it's faked like crazy. Real physics aren't that fun—if I fall 12 feet, I break my leg. Real AI will headshot you every time. So you want the game to look intelligent and realistic while still having fun beating it. We need to enable as much realism as the developers want, then allow them the fine control to step sideways. An example of that in graphics is colorizing and brightening, such as in the TV show *Pushing Daisies*, where scenes are filled with pastels, or have huge contrast, or their hue changes to emphasize a mood. The colors are completely unrealistic and yet we not only accept them, they tell us things without conscious input. And it all relies on exquisitely fine control over the rendering. That is what Larrabee will provide to developers: new techniques for enhancing the visceral feel of a game. ▪

Tom's blog is found at:
www.eelpi.gotdns.org/blog.wiki.html

RESOURCES

Intel® Software

Intel's heightened focus on visual computing and graphics processing is complemented by software development products, graphics chipsets, professional services, technical expertise, and developer-oriented resources. Keep up with the latest activities of the Intel® Visual Adrenaline Developer Program at www.intel.com/software/visualadrenaline.

Explore topics from this issue of *Visual Adrenaline* further:

*Wolfenstein**: Rebuilt from the Ground Up for Today's Hardware

To view the *Wolfenstein** video, go to: www.gametrailers.com/game/9183.html

For more about id's *Wolfenstein 3D** go to: www.activision.com/index.html#gamepage|en_US|gameId:Wolf3D&brandId:Wolfenstein

For information about id's *Return to Castle Wolfenstein** go to: www.activision.com/index.html#gamepage|en_US|gameId:ReturnToCastleWolf&brandId:Wolfenstein

To learn about Raven Software Games go to: http://ravensoft.com/Portals/0/games.aspx

Havok tools can be found at: www.havok.com

Indie Game Demo Developers Speak Out on Their Victories

PillowFort: http://goo.pillowfortgames.com/

Tandem Games: www.tandemgames.com/

Frogames: www.frogames.com/

To download any of these winning game demos, go to: http://software.intel.com/en-us/articles/spotlight-on-the-2008-intel-game-demo-contest-winners/

*Empire: Total War** Takes to the Waves

See the official Web site for *Empire: Total War* at: www.totalwar.com/empire

Learn about The Creative Assembly at: www.creative-assembly.co.uk

Learn more about SEGA at: www.sega.com/games/empire-total-war

To purchase and download popular games, go to: http://store.steampowered.com/app/10500

Composing Audio for Video Games: Art— Enabled by Science and Cakewalk Software

For highlights of recent activities at Cakewalk and more artist stories, visit the Cakewalk Blog: http://blog.cakewalk.com

For a guided tour through Cakewalk SONAR* 8, view the Webinar on YouTube*: www.youtube.com/watch?v=Lg_1dDUBry4

To experience the capabilities of SONAR 8 Producer, download a 30-day trial version from: www.cakewalk.com/support/kb/kb20081217.asp

For more information about the history of multi-threading in Cakewalk applications, download the Cakewalk Technology White Paper, *Multiprocessing in SONAR 3.1*: www.cakewalk.com/DevXchange/Multiprocessing_in_SONAR_3.pdf

For more information on the advantages of 64-bit processing in Cakewalk's SONAR, download the Cakewalk Technology White Paper *The Benefits of Modern CPU Architectures for Digital Audio Applications*: www.cakewalk.com/Press/Cakewalk_White%20Paper_Benefits_of_Modern_CPU_Architectures_for_Digital_Audio_Applications.pdf

For a video celebrating the 20-year history of Cakewalk, visit: www.youtube.com/watch?v=jn7o3cpYzhY

Benchmarks that show the CPU performance gains for typical tasks running SONAR 8 (compared to previous versions) can be found at: www.cakewalk.com/Products/SONAR/English/benchmark.asp

- Subscribe to the *Intel® Visual Adrenaline* magazine: www.intel.com/software/visualadrenaline
- To learn more about becoming a member visit: www.intel.com/partner/visualcomputing
- To find resources in our Developer Community, visit: www.intel.com/software/graphics
- To learn more about Intel® Graphics Performance Analyzers, visit: www.intel.com/software/gpa
- Visit Intel's new Media Resources site at: www.intel.com/software/media

Stay current on the latest software developments in Visual Computing at: www.intel.com/software/visualadrenaline