CONSUMER ELECTRONICS, DEVELOPERS

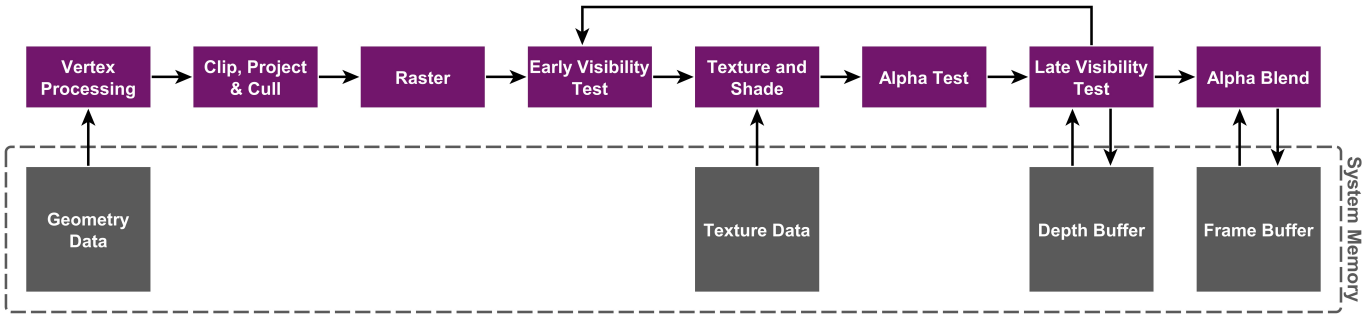
A look at the PowerVR graphics architecture: Tile-based rendering

02 APRIL 2015 | KRISTOF BEETS | 16 COMMENTS

I'm fond of telling the story about why I joined Imagination. It goes along the lines of: despite offers to go work on graphics in much sunnier climes, I took the job working on PowerVR Graphics here in distinctly un-sunny Britain because I was really interested in how ***Tile-Based Deferred Rendering (TBDR)*** could work in practice. My graphics career to-date had been mostly focused on the conceptually simpler ***Immediate Mode Renderers (IMRs)*** of the day – mostly GeForces and Radeons.

And no offence to the folks who designed said GeForces and Radeons – a few of whom I am friends with and many more I know quite well, but the front-end architecture of a modern discrete IMR GPU isn't the most exciting thing in the world. Designed around having plenty of

dedicated bandwidth, those GPUs go about the job of painting pixels in a reasonably inefficient way, but one that's conceptually simple and manifests itself in silicon in a similarly simple way, which makes it relatively easy for the GPU architect to design, spec and have built by the hardware team.



Immediate Mode Rendering at work

With an IMR you send work to the GPU and it gets drawn straight away. There's little connection to what else has already been drawn, or will be drawn in the future. You send triangles, you shade them. You rasterise them into pixels, you shade those. You send the rendered pixels to the screen. Triangles in, pixels out, job done! But, crucially the job is done with no context of what's already happened, or what might happen in the future.

PowerVR GPUs are about as different as they come in that respect, and it's that which made me take the job here, to figure out how PowerVR's architects, hardware teams and software folks had made TBDR actually work in real products. My instinct was that TBDRs would be too complex to build so that they'd work well and actually provide a benefit. I had a chance to figure it out and five years later I'm still here, helping figure out how we'll evolve it in the future, along with the rest of the GPU's microarchitecture.

As far as the graphics programmer is concerned, PowerVR still looks like *triangles in, pixels out, job done*. But under the hood something much more exciting is happening. And while the exciting part put me in this chair so I could write about it 5 years later, crucially it's also that other good E word: efficient!

It always starts with the classic TBDR vs. IMR debate

To help understand why, let's keep talking about IMRs. One of the biggest things stopping a desktop-class IMR scaling down to fit the power,

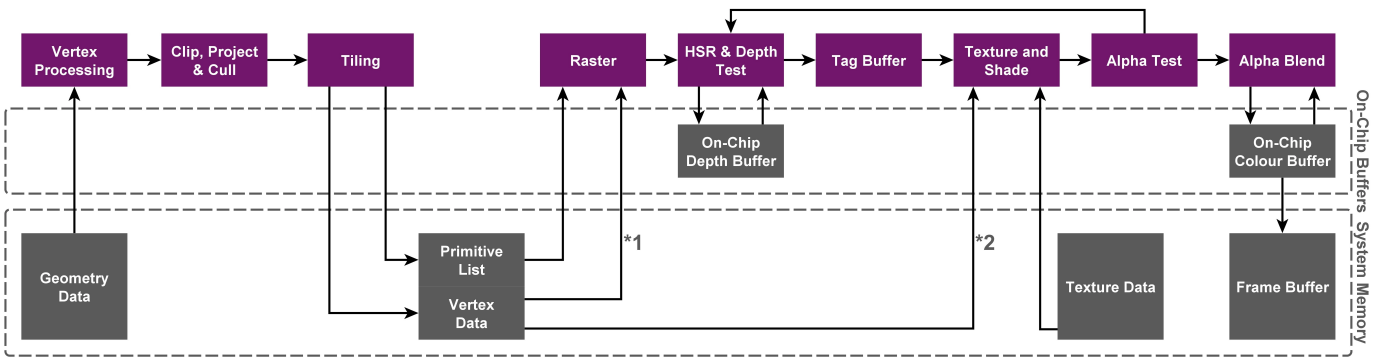
performance and area budgets of modern embedded application processors is bandwidth. It's such a scarce resource, even in high-end processors – mostly because of power, area, wiring and packaging limitations, among other things – that you ***really*** need to use it as efficiently as possible.

IMRs don't do that very well, especially when pixel shading. Remember that there are usually a great many more pixels being rendered than the vertices used to build triangles. On top of that, with an IMR pixels are often still shaded despite never being visible on the screen, and that costs large amounts of precious bandwidth and power. Here's why.

Textures for those pixels need to be sampled, and those pixels need to be written out to memory – and often read back in and written out again! – before being drawn on the screen. While all modern IMRs have means in hardware to try and avoid some of that redundant work, say one building in the background being completely obscured by one drawn closer to you, there are things the application developer can do to effectively disable those mechanisms, such as always drawing the building in the background first.

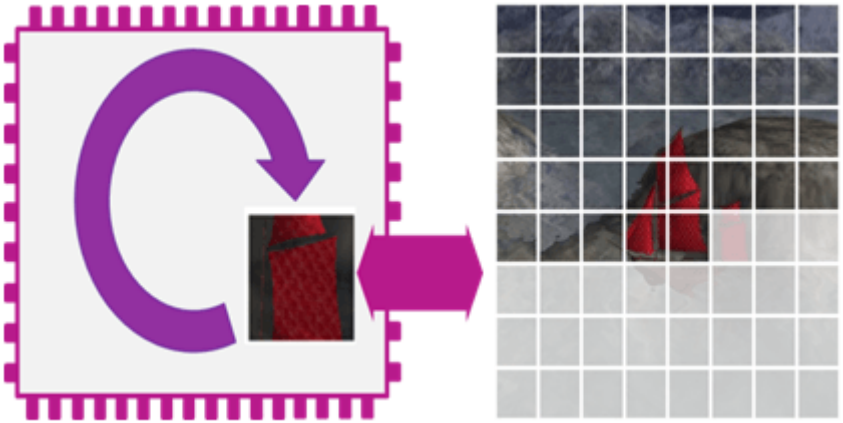
In our architecture it doesn't really matter how the application developer draws what's on the screen. There are exceptions for non-opaque geometry, which the developer still needs to manage, but otherwise we're submission order independent. That capability is something we've had in our hardware since before we were ever an IP company and still made our own standalone PC and console GPUs. You could draw the building in the background first, then the one in the foreground on top, and we'll never perform pixel shading for the first one, unlike an IMR.

We effectively sort all of the opaque geometry in the GPU, regardless of how and when it was submitted by the application, to figure out the top-most triangles. Sure, if a developer perfectly sorts their geometry then an IMR can get much closer to our efficiency, but that's not the common case by any means.



Think again about all the work that's saving, especially for modern content: for ***every pixel*** shaded there are going to be a non-trivial amount of texture lookups for various things, dozens and sometimes hundreds of ALU cycles spent to run computation on that texture data in order to apply the right effects, which often means writing the pixel out to an intermediate surface to be read back in again in a further rendering pass, and then the pixel needs to be stored in memory at the end of shading, so it can be displayed on screen.

And that's just one optimisation that we have. So even though we've avoided processing completely occluded geometry, there's still bandwidth saving work we can do at the pixel processing stage. Because we split the screen up into tiles, where we figure out all of the geometry that contributes to the tile so we only process what we need to, and we know exactly how big the tile is (currently 32×32 pixels, but it's been smaller and even non-square in prior designs), we can build enough on-chip storage to process a few of those tiles at a time, without having to use storage in external memory again until we've finished and want to write the final pixels out.



PowerVR GPUs split the screen into tiles

There are secondary benefits to working on screen, region at a time; benefits that other GPUs take advantage of too: because it's highly likely that a pixel on the screen will share some data with its immediate neighbours, it's likely that when we move on to processing the neighbouring pixels that we've fetched the data into cache and don't have to wait for another set of external memory accesses, saving bandwidth again. It's a classic exploitation of spatial locality that's present in a lot of modern 3D rendering.

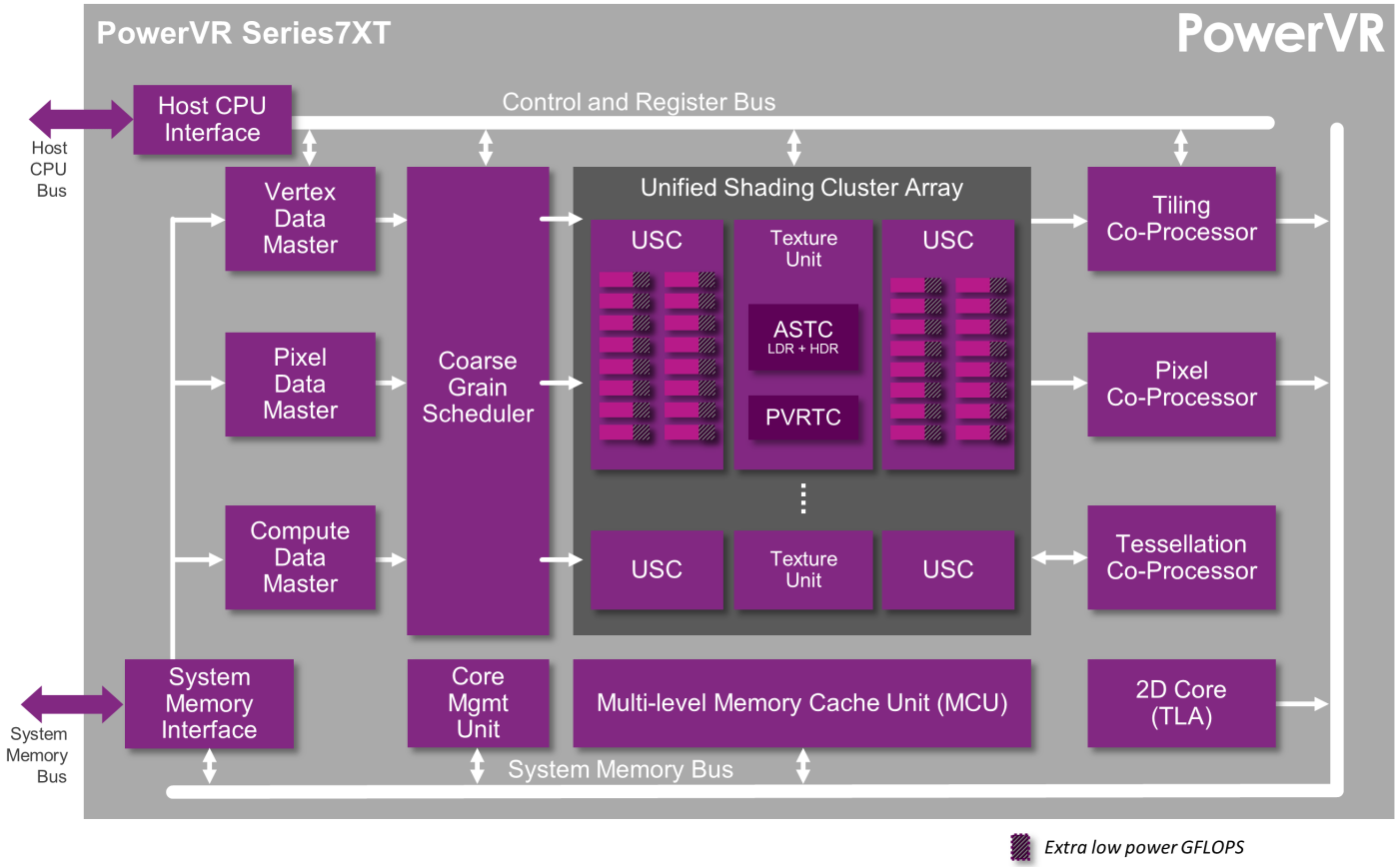
So that's the top-level view of the biggest benefits of a TBDR versus an IMR in terms of processing and (especially) bandwidth efficiency. But

how does it actually work in hardware? If you're not too hardware inclined you can stop here! If you go no further, you'll still have understood the big top-level benefits of how we go about making best use of the available and very precious bandwidth, throughout rendering on in embedded, low-power systems.

How TBDR works in hardware

For those interested in how things happen in the hardware, let's talk about the tiler in context of a modern Rogue GPU.

A 3D graphics application starts by telling us where its geometry is in memory, so we ask the GPU to fetch it and perform vertex shading. We have blocks in our GPUs that are responsible for the non-programmable steps of each kind of task type, called ***the data masters***. They do a bunch of different things on behalf of ***the Universal Shading Cluster or USC*** (our shading core) to do the fixed function bits of any workload, including fetch data from memory. So because we're vertex shading, it's ***the vertex data master (VDM)*** that gets involved at this point, to fetch the vertex data from memory based on information provided by the driver.



PowerVR Series7XT is the latest family of Rogue GPUs

The data could be stored in memory as lines, triangles or points. It could be indexed or non-indexed. There are associated shader programs and accompanying data for those programs. The VDM fetches whatever's needed, using another couple of internally-programmable blocks to help, and emits it all to the USC for vertex shading. The USC runs the shader program and the output vertices are stored on-chip.

They're then consumed by hardware that performs primitive assembly, certain kinds of culling, and then clipping. If the geometry is back-facing or can be determined to be completely off the screen, it's culled. All of the remaining on-screen front-facing geometry is sent to be clipped. There's a fast path here for geometry that doesn't intersect a clip plane, to let it make onwards progress with no extra processing bar the intersection test. If the geometry intersects with a plane, the clipper generates new geometry so that passed-on vertices are fully on-screen (even though they might be right at the edges). The clipper can do some other cool things, but they're not too relevant for a big picture explanation like this.

Then we're on to where a lot of the magic happens, compared to IMRs. We're obviously aiming to run computation in multiple phases in the hardware, to maximise efficiency and occupancy: one front-end phase to figure out what's going on with shaded geometry and bin it into tiles, then one phase to consume that binned data, rasterise it and pass it on for pixel shading and final write-out. To keep things as efficient as possible that intermediate acceleration structure between the two main phases has to be as optimal as we can make it.

Clearly it's a bandwidth cost to create it and read it back, one which our competitors like to pick on when it comes to a competitive advantage they have over us. And it's true; an IMR doesn't have to deal with it. But given the bandwidth savings we have in our processing model, creating that acceleration structure – which we call ***the Parameter Buffer (PB)*** – before feeding it to our trick rasteriser, we still end up with a huge bandwidth advantage in typical rendering situations, especially complex game-like scenes.

So how do we generate the PB? The clipper outputs a stream of primitives and render target IDs into memory, grouped by render target. Think of it as a container around collections of related geometry. The relationship is critical: we don't want to read it later and not consume a majority of the data that's inside, since that'd be wasteful. The output at this stage is the main data structure stored in the PB. We then compress the memory, and in the general case it always compresses very well, so we save quite a lot of PB creation bandwidth just from that step alone.

The concept of tiling

Now for the bit that most people sort of understand about our hardware architecture: tiling. The tiling engine has one main job: output some data that marks out a tiled region, some associated state, and a set of pointers to the geometry that contributes to that region. We also

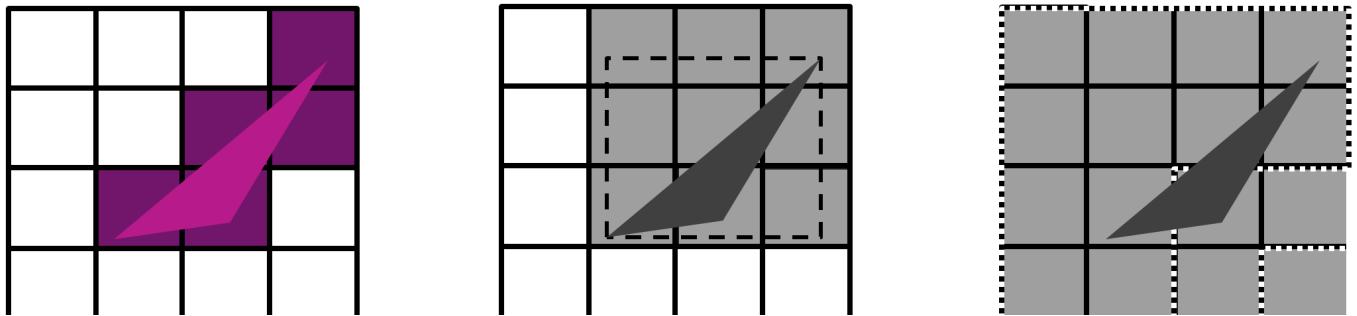
store masks just in case a primitive doesn't actually contribute to the region, but is stored in memory anyway. That lets us save some bandwidth and processing for that geometry, because it doesn't contribute to the tile.

We call the resulting data structure a primitive list. If you've ever consumed any of our developer documentation, you'll have seen mention of primitive lists as the intermediate data structure between the front-end phase and the pixel processing phase. Next, some more magic that's specific to the PowerVR way of doing things.

Imagine you were tasked with building this bit of the architecture yourself, where you had to determine what regions need to be rasterised for a given set of geometries. There's one obvious algorithm you could choose: the bounding box. Draw a box around the triangle that covers its extents, and whatever tiles that box touches are the ones you rasterise for that triangle. That falls down pretty quickly though, efficiency wise.

Imagine a fairly long and thin triangle drawn across the screen in any orientation. You can quickly picture that the bounding box for that triangle is going to lie over tiles that the triangle doesn't actually touch. So when you rasterise, you're going to generate work for your shading core, but where nothing is actually going to happen in terms of a contribution to the screen.

Instead, we have an algorithm baked into the hardware which we call ***perfect tiling***. It works as you'd expect: we only generate tile lists where the geometry actually covers some area in the tile. It's one of the most optimised and most efficient parts of the design. The perfect tiling engine generates that perfect list of tiles for a given set of geometry.



PowerVR perfect tiling vs. bounding box or hierarchical tiling

That tile information plus the primitive lists are packed into the PB as efficiently as we can, and that's conceptually pretty much it. In reality there's a heck of a lot that still happens here in the hardware at the back-end phase of tiling, to fill the PB and organise and marshal the actual memory accesses for the external memory writes, but in terms of functionality to wrap your head around, we're pretty much done.

That front-end hardware architecture for us is really where a really big chuck of the efficiency gains can be found in a modern PowerVR GPU, compared to some of our competition. Surprisingly to those who find out, it's part of the hardware architecture that's not actually that different, at least at the top-level, between Rogue and SGX. While we completely redesigned the shader core for Rogue, the front-end architecture actually bears a strong resemblance to the one you'll find in the later generation SGX GPU IPs. It works and it works very well.

And now that I'm done explaining the tiling part of our TBDR, it's a good excuse to stop! I'll come back to the deferred rendering part in a future blog post, so stay tuned.

developers | Embedded Graphics | Game Development | GPU | Mobile | PowerVR | PowerVR developers | Rogue

Kristof Beets

Kristof Beets is the senior director of product management for PowerVR Graphics at Imagination Technologies where he drives the product roadmaps to

ensure alignment with market requirements. Prior to this, he was part of the business development group and before this, he led the in-house demo development team and the competitive analysis team. His engineering background includes work on SDKs and tools for both PC and mobile products as a

member of the PowerVR Developer Relations Team. His work has been published in many guides game and graphics programmers, such as Shader X2, X5 and X6, ARM IQ Magazine, and online by the Khronos Group, Beyond3D.com and 3Dfx Interactive. Kristof has a background in electrical engineering and received a Master's degree in artificial intelligence. He has spoken at GDC, SIGGRAPH, Embedded Technology, MWC and many other conferences.

16 thoughts on "A look at the PowerVR graphics architecture: Tile-based rendering"

Chocolate Fudge

July 30, 2015 at 4:06 am

Great article Rys. I've always been interested in graphics programming, but never was able to kick-start my journey to getting deeper into it and I simply lost track of the desire over the years as I shifted to developing general platform tools. This article rejuvenated my goal!

Reply

Goblinit

July 17, 2015 at 5:13 pm

Hi, Rys!
Thanks a lot for this post! You know this is very good because it explains some hardware things more clear than anywhere else. I've been looking for some kind of simple explanation about TBDR for years, and you posted it at last. Thanks!! When will be the next stop? 🙂

Excuse my bad English if it is, Rys. Just wanted to ask you one simple question (maybe not so simple though…). Heard about any old-gamer communitys today such as VOGONS (https://www.vogons.org)? There we are not only playing old games, but trying to test old hardware that was so hard to get those days, you know.
So the question will be: I had a graphics card for PC with PowerVR PCX2 chip onboard in late 90-s. Now it's gone and hard to find again. Besides I'm interested in one thing: that chip (PCX2) supported some kind of graphics API called PowerSGL. There were some computer games, that were using it (ex. Unreal 1). We already have found a lot of them on our forum (https://www.vogons.org/viewtopic.php?p=279818#p279818). Can you explain what unique features or maybe rendering methods were suggested by this PowerSGL api?

Reply

richardw
April 7, 2015 at 1:00 pm

I'm a little puzzled as to why PowerVR graphics isn't more successful in the Android marketplace given the efficiencies of TBDR you explain. Is it that other approaches used by IMR architectures (as suggested by John Brooks) compensate? I do of course know about the success you have with Apple, which is another reason I'm so puzzled about the Android market performance.

Reply

Alexandru Voica
April 7, 2015 at 3:01 pm

Hi,
When it comes to Android, there are multiple CPU and GPU architectures to choose from. If you look at recent design wins, PowerVR Rogue GPUs can be found in more than ten chips used in Android devices (smartphones, tablets, smart TVs, etc.):

– Actions Semiconductor ATM9009 (part of the new Falcon series) (64-bit quad-core CPU, PowerVR G6230 GPU) for tablets and OTT set-top boxes
– Allwinner UltraOcta A80 (32-bit octa-core CPU, PowerVR G6230 GPU) for tablets, set-top boxes, portable game consoles and many more
– Intel Atom Z3460/Z3480 Merrifield (64-bit dual-core CPU, PowerVR G6400 GPU) and Z3560/Z3580 Moorefield (64-bit quad-core CPU, PowerVR G6430 GPU) for smartphones and tablets
– LG Nuclun (32-bit octa-core CPU, PowerVR G6430 GPU) for phablets
– MediaTek MT8135 (32-bit quad-core CPU, PowerVR G6200 GPU) and MT8173 (64-bit quad-core CPU, PowerVR GX6250 GPU) for tablets, MT6595 (32-bit octa-core CPU, PowerVR G6200 GPU) and MT6795 (64-bit octa-core CPU, PowerVR G6200 GPU) for smartphones
– Rockchip RK3368 (64-bit octa-core CPU, PowerVR G6110 GPU) for tablets and OTT set-top boxes
Regards,
Alex.

Reply

richardw
April 7, 2015 at 3:33 pm

Thanks Alex, this is useful. However why is it that, from the list, Intel appears to be adopting Mali and it is rumoured that Mediatek will be licensing AMD graphics? It just seems to me that PowerVR's position is weakening despite its historic market dominance, I know huge market share has been lost over the past couple of years without any real sign that it is being rebuilt.

Reply

Alexandru Voica
April 7, 2015 at 3:41 pm

I cannot address rumors and speculation nor speak on behalf of our customers.
Regards,
Alex.

Reply

richardw
April 7, 2015 at 6:53 pm

> Of course. I only asked as those customers were a couple of those you quoted. Going back to my original question of TBDR vs IMR, I think it is of wide interest to technologists and investors to know IMG's assessment of this, taking into account the techniques used with IMR these days as mentioned by John Brooks

Alexandru Voica
April 8, 2015 at 10:04 am

> I think there is some confusion in the terminology used in software vs. hardware. Simply because a GPU supports software techniques like deferred shading doesn't mean it is architecturally close(r) to TBDR or PowerVR.
> Like Rys states in the comment below, PowerVR offers significant benefits in bandwidth reduction for a lot of the common software workloads – something that IMRs have not had to deal with in the past since they have been designed for desktop first (where memory bandwidth and power consumption have not been historically relevant).

LDM
April 7, 2015 at 8:17 pm

Hi Aex,
indeed this year has been very successful for the PowerVR wins in many devices.
However I do understand richardw concern of the increasing of competition around in particular in some supplier where historically Img was the main supplier: Intel and Samsung; they seem moving to Mali GPU.
Having said that it looks Img has a predominant market in the low-middle range GPU support with the G6400-6430 GPU comparing to your direct competitors. The latter, instead, seem to win more high-end solution devices such the latest Mali 760 etc.
In this sense I hope to see more high-end series 6 during this year (6xt?) in more devices, apart of course your main lead supplier Apple, in the Android markets.
Fantastic article b the way!!
L

Reply

Alexandru Voica
April 8, 2015 at 10:07 am

Some companies prefer to source GPUs from different suppliers; examples include Allwinner, Rockchip, MediaTek, Samsung and others. Other companies also have in-house solutions that they use in certain applications (e.g. Intel).
Having multiple players in the market spurs innovation and healthy growth. This is valid for GPU, CPU or any other dedicated processor.

Reply

Reply

LDM

May 7, 2015 at 10:06 pm

I know Alex and it's fair enough what you said.
But reality is that this year you guys haven't placed any high end series 6 in any devices. This is the reality and this makes reason of concern.
Just last news: Mediatek has presented the new Helio x20 soc. It has the new Mali T880. I hoped there was instead some Series 6XT or even series 7. It wasn't and…so be it..

Hideki Ito

April 3, 2015 at 12:47 am

i remember reading about Power VR back in the mid 90s in Next Generation magazine. Loved the DreamCast, and it's great to see the PowerVR technology still around!

Reply

John Brooks

April 2, 2015 at 11:43 pm

As a longtime graphics engine programmer on both TBDR and IMR GPU's, I'm interested in how TBDR compares to IMR for engines

using modern rendering techniques and GPUs. Specifically, I wonder if the combination of Z-prepass (for IMR), deferred lighting/shading, and on-chip render target memory (hi-Z and potentially entire frame buffers in embedded ram) allows IMR GPUs to obtain many of the bandwidth savings traditionally achieved by TBDR. It seems like IMR may have the advantage of streaming source textures through the GPU once (assuming scene-wide sort-by-texture) where TBDR runs the risk of texture reload due to cache eviction between tiles. Is it possible to control the order in which tiles are rendered to avoid/reduce cache thrashing? It would be great if you could touch on some of these topics in a future post.

Reply

Rys Sommefeldt
April 8, 2015 at 12:20 am

Of course it's possible to engineer renders that have very little bandwidth saving on a TBDR, but in general the technique is a clear win. If you then combine our technology with ways to programmatically access the per-pixel tile memory, you can continue to save even more bandwidth than an IMR using those same rendering algorithms designed around modern desktop IMRs. For example, with deferred shading it's potentially possible to generate the entire G-buffer on-chip, with no external bandwidth needed for storing that buffer, depending on the size of it.

So it's renderer dependent as to what kind of saving you get, but it's hard to have no saving at all.
As for tile rendering order, there's no programatic control at the moment, but it's an area of active research for the future.

Reply

MrsTech
April 2, 2015 at 5:43 pm

Looking forward to reading more about your technology.

Reply

Please leave a comment below

Comment policy: We love comments and appreciate the time that readers spend to share ideas and give feedback. However, all comments are manually moderated and those deemed to be spam or solely promotional will be deleted. We respect your privacy and will not publish your personal details.

Name *

Email *

Save my name and email in this browser for the next time I comment.

Post Comment

Blog Contact

If you have any enquiries regarding any of our blog posts, please contact:

United Kingdom

benny.har-even@imgtec.com

Tel: +44 (0)1923 260 511

Search by Tag

Search for posts by tag.

Select a Tag

Search by Author

Search for posts by one of our authors.

Select an Author

Featured Posts

Learning From Home: 5 Free Technology Webinars to Keep Yourself Up to Date

27 MAY 2020 BENNY HAR-EVEN

How AI is conducting the future of music technology

05 FEB 2019 PAUL WEIR

Separating the wheat from the chaff in embedded AI with PowerVR Series3NX

Separating the wheat from the chaff in embedded AI with PowerVR Series3NX

24 JAN 2019 BENNY HAR-EVEN

The ultimate embedded GPUs for the latest applications
06 DEC 2018 BENNY HAR-EVEN

Imagination Technologies: the ray tracing pioneers
10 OCT 2018 BENNY HAR-EVEN

Amazon Lights up its Fire TV Stick 4K with PowerVR
05 OCT 2018 BENNY HAR-EVEN

Neural networks – a guide for my mom
25 SEP 2017 JEN BERNIER

Face detection and identification using OpenCL on PowerVR GPUs
12 SEP 2017 ASHLEY SMITH

Popular Posts

Why you really should be using mipmapping in your graphics applications

PowerVR SGX544, a modern GPU for today's leading platforms

A look at the PowerVR graphics architecture: Tile-based rendering

Understanding OpenGL ES: Multi-thread and multi-window rendering

Understanding OpenGL ES: Multi-thread and multi-window rendering

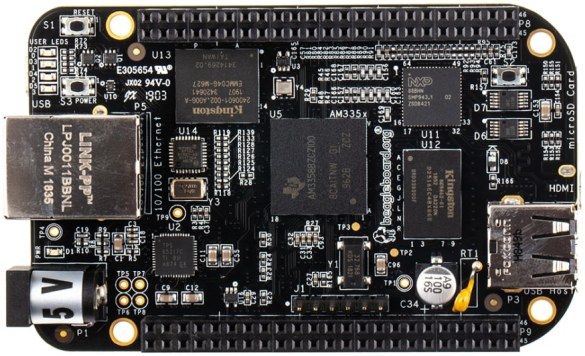
Implementing fast, ray traced soft shadows in a game engine

A consumer's guide to graphics benchmarks

Unreal Engine and the ray tracing revelation

Creating Unreal Engine 360° panoramas the easy way with ray tracing

Related blog articles



Fun with PowerVR and the BeagleBone Black: Low-Cost Development Made Easy

Why you should be running your iOS apps on your new Apple M1 laptop

February 25, 2021

Development boards are cool and the BeagleBone® Black (BBB) is one of the more interesting ones around. This widely available tiny board costs around £35 and will boot Linux is only 10 seconds so anyone interested in development can get stuck in quickly. The Introduction to Mobile Graphics course has been recently revamped for 2020 for the Imagination's University Programme and the widely available, low-cost BBB is an ideal platform for student teaching and exercises based on OpenGL® ES2.0, instead of an expensive standard PC.

Read More »

February 23, 2021

Towards the end of last year, Apple released the latest version of its Apple MacBook Pro and Macbook Air laptops. This release was notable as with these brand-new laptops Apple made a significant change – the processor inside was based on its own M1 chip rather than the Intel architecture that it had been using since 2006. Since its release, the Apple M1 has been widely hailed for its performance, outstripping Intel in all the major benchmarks and all in a cool, quiet package with low power consumption.

Read More »

The Android Invasion: Imagination GPU IP buddies up with Google-powered devices

December 14, 2020

Google Android continues to have the lion share of the mobile market, powering around 75% of all smartphones and tablets, making it the most used operating system in the world. Imagination's PowerVR architecture-based IP and the Android OS are bedfellows, with a host of devices based on Android coming to market all the time. Here we list a few that have appeared in Q4 2020.

Read More »

Connect

Sign up to receive the latest news and product updates from Imagination straight to your inbox.

First name *

Last name

Email *

Enter your email address

Region *

Please Select

Imagination Technologies is committed to protecting and respecting your privacy, and we'll only use your personal information to administer your account and to provide the products and services you requested from us. From time to time, we would like to contact you about our products and services, as well as other content that may be of interest to you. If you consent to us contacting you for this purpose, please tick below to say how you would like us to contact you:

I agree to receive other communications from Imagination Technologies. *

In order to provide you the content requested, we need to store and process your personal data. If you consent to us storing your personal data for this purpose, please tick the checkbox below.

I agree to allow Imagination Technologies to store and process my personal data. *

You can unsubscribe from these communications at any time. For more information on how to unsubscribe, our privacy practices, and how we are committed to protecting and respecting your privacy, please review our Privacy Policy.

Sign Up

Processors

Graphics

AI

CPU

Ray Tracing

Ethernet

Design Optimization Kit

Product Finder

Markets

Automotive

Consumer

Desktop

Mobile

Universities 大学

News & Events

Blog

News

Press Releases

Events

Webinars

Presentations

Developers

Developers

PowerVR SDK and Tools

Developer Downloads

Developer Documentation

Developer Support

Developer Forum

Company

About Us

Why Work With Us

Contact Us

Careers

Leadership Team

Corporate Responsibility

Downloads and Documentation

Resources

Modern Slavery Act