

Enlighten Research, past, present, future

Sam Martin
Head of Technology

London Graphics Seminar 2011



Geomerics

Hello!

Agenda

- Enlighten
- Research highlights - Past
 - › Observations on accuracy
 - › Lighting research with UCL
- Research highlights - Present
 - › Enlighten CUDA
- Future thoughts
 - › Enlighten as a pipeline stage
 - › Future graphics pipelines with Enlighten
- QA

London Graphics Seminar 2011

Geomerics

This talk will be about some highlights of the research done at Geomerics and in collaboration with UCL, past present and future.

It isn't intended to be a comprehensive account of our research. Just a few specifics, some details on our current research and then I will allow to digress briefly into speculation at the end.

I have selected topics that I feel are in some way understated but important, so the main takeaway I'm aiming for it to offer some food for thought from the work we and others have done, and point out where I think further research could be beneficial.

But first I will give a very brief overview of Enlighten so you are aware of its structure.

Enlighten

- Enlighten is a revolutionary lighting technology available for PC, Xbox 360 and PS3
- In use in over 15 titles across the globe
 - › Mixture of UE3 and stand-alone SDK
- Core runtime is mature
- Underlying principles and workflows fixed



DUST 514



EVE®

London Graphics Seminar 2011

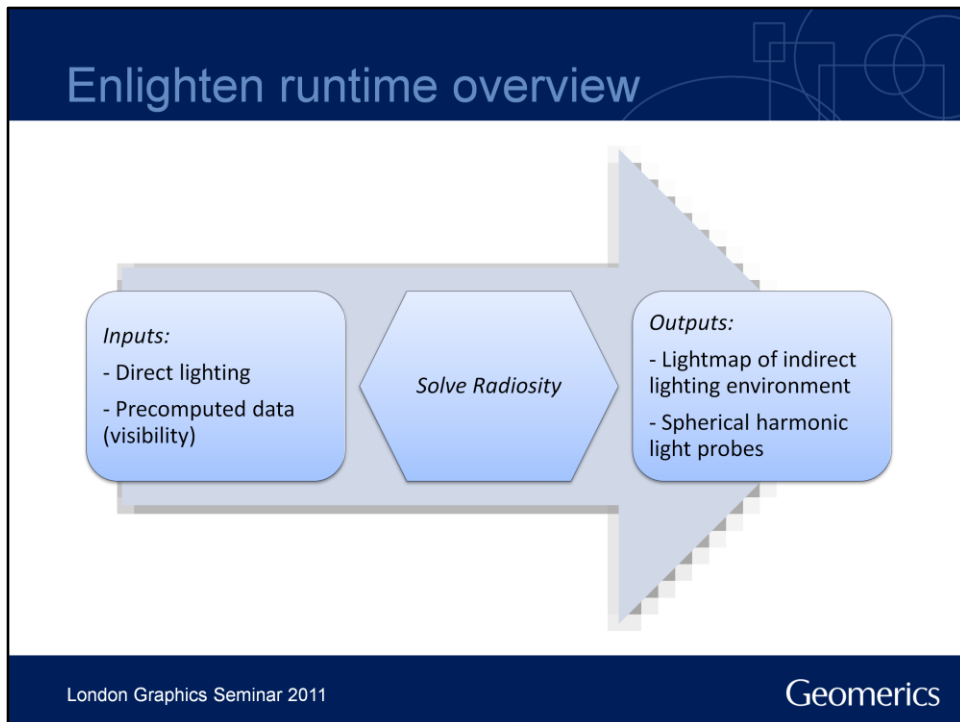
Geomerics

What is Enlighten?

Enlighten is lighting middleware. It computes indirect lighting in real time, primarily for today's console games.

It's been in development since 2006 and is now mature and we will see the first shipping games this year

Shipping games this year include Eve Online, Battlefield 3 and another major unannounced franchise.



Here is a quick overview of what the Enlighten runtime for generating indirect lighting looks like.

I will keep this brief. For further details please refer to my talk, “A Real time Radiosity Architecture for Video Games”, done together with Per Einarsson from EA DICE at SIGGRAPH last year (2010).

You can think of the Enlighten runtime as a **lightmap generation function**:

- Think of it as a function mapping lights and precomputed data to bounced lighting
- It runs asynchronously to graphics pipeline on the CPU on regular PCs and consoles
- The direct lighting description on CPU is typically a list of typical spot, point, and directional light, plus Enlighten-specific environment light (like a HDRI env map) and area lights (think: emissive materials) .
- This direct lighting description is actually used to generate samples of incident radiance. This sample input is actually more fundamental and possible to provide directly, as are other variations.
- The (constant) precomputed data is generated in a typically 10-minute offline step based on geometry only. It essentially captures visibility info and magic ☺.



(Apologies for the aspect ratio issues! I borrowed these images from my SIGGRAPH talk.)

This is the traditional direct lighting. It's just a simple cascade shadow map based directional light.

This is just an example, the choice of light and its implementation is essentially arbitrary.

Note that although there is a blue sky, there is no direct *environment* lighting in this shot. Environment lighting (and other area lights) are handled entirely within Enlighten.



This is the corresponding input to Enlighten, after conversion to the point sampled input.

The core runtime component of Enlighten maps this point sampled description of the lighting, to a lightmap or lightprobe output.

Anything you can express in these terms is valid input to Enlighten.

Sidenote: You may note that as well as the directional light, there is also radiosity lighting in the point sampled data. This is the previous lightmap output being fed to enlighten as an input light to generate a second bounce. This is how we generate multiple bounces. (Again, see the SIGGRAPH talk for further details)



This is the output from Enlighten, together with indirect specular and relighting of detailed geometry and normal maps, but without the direct sunlight or textures.

This is Enlighten's contribution to the final image.



And here we see the final composite.

This was a brief introduction to our runtime radiosity pipeline.

Research highlights - Past

- Observations on accuracy in lighting
- Two highlights from TSB funded research
 - › Perceptual influence of approximate visibility
 - › Radiosity and Ambient Occlusion

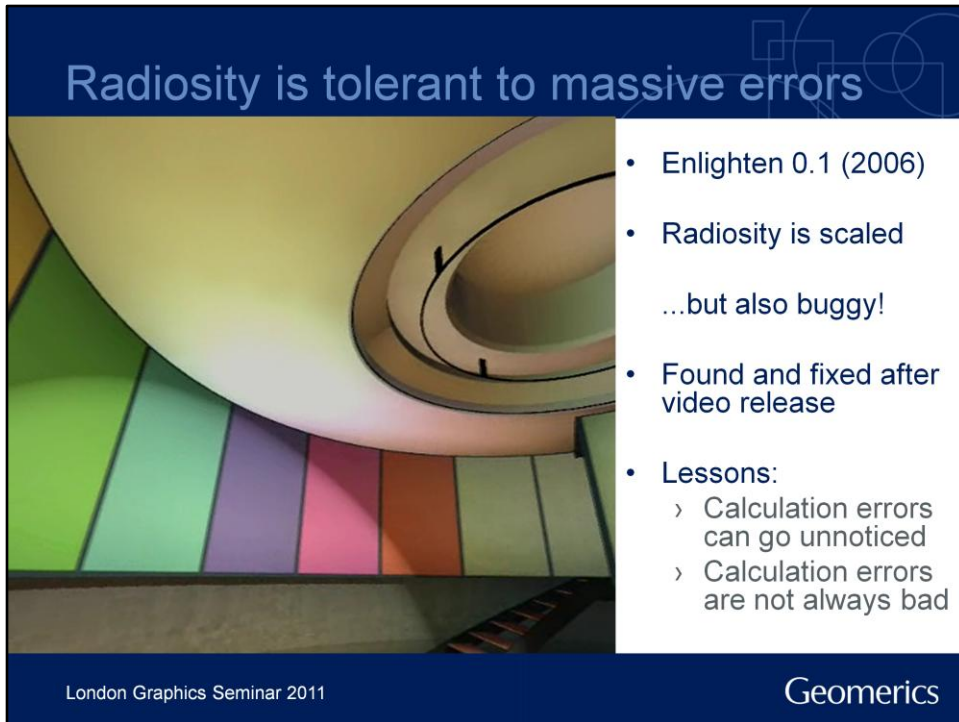
London Graphics Seminar 2011

Geomerics

So, now I've quickly described Enlighten, I'd like to pick on a few observations we and UCL made during the development of Enlighten.

They are select topics I feel haven't been "discussed enough" and still have some research legs in them.

Radiosity is tolerant to massive errors



- Enlighten 0.1 (2006)
- Radiosity is scaled
...but also buggy!
- Found and fixed after video release
- Lessons:
 - › Calculation errors can go unnoticed
 - › Calculation errors are not always bad

London Graphics Seminar 2011

Geomerics

Ok, first I have to put my hands up.

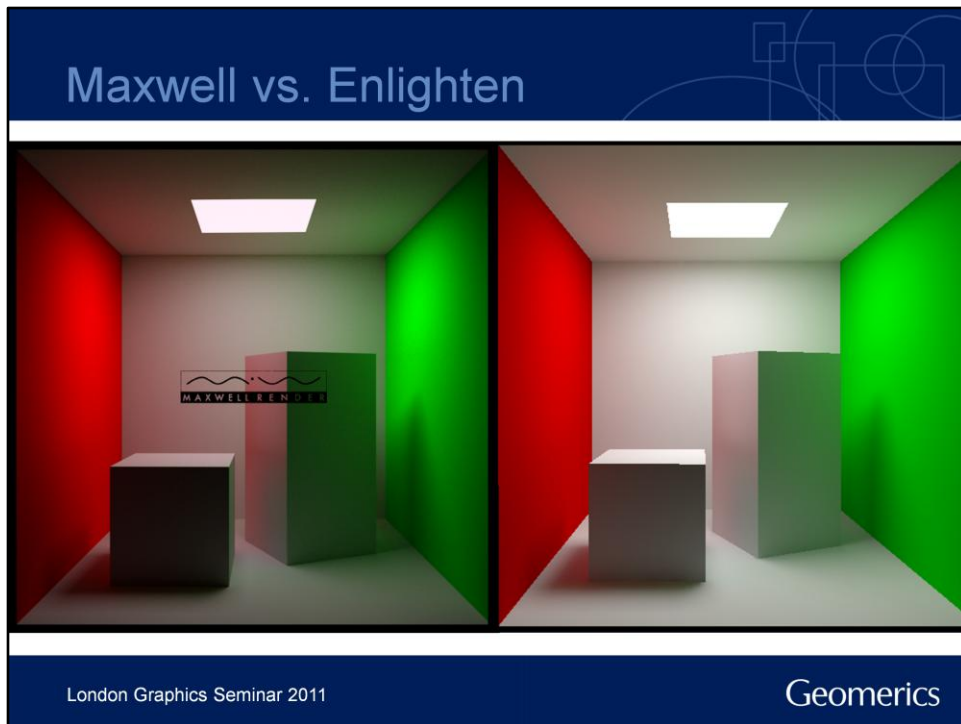
This is an image from the very first video we released of our first prototype of what would become Enlighten, pre- any release way back in 2006. This is when the company was just 4 employees and the product didn't yet have a name.

The image shows a room lit with one spotlight (shining on the lower left), and a nice strip of (Enlighten-generated) multi-coloured bounce lighting across the ceiling. We had boosted the radiosity overall to make it stand out, but that was it.

However, after the video went out, we discovered a fairly major bug. I forget the exact details, but it was failing to integrate a very significant chunk of the hemisphere in a very structured "this never should have worked" kind of way.

But it *did* work. Or at least, we couldn't see the error. In fact, when we did find and fix it, the result looks less attractive and we spent some time improving the tech till we could re-capture the same effect.

This, and other observations, were an early lesson in how inaccurate your radiosity integrations can be – under the right conditions. Sadly the reverse is also true: Very minor discrepancies can be very obvious. A lot of work in Enlighten has been in getting this balance correct.



In case that admission mistakenly gives you the impression Enlighten isn't "accurate", here is a comparison with Maxwell (a physically based, computationally heavy third party tool) from a build of Enlighten2 last year some time.

Maxwell is on the left, Enlighten is on the right.

There are no huge bugs in Enlighten this time! But we are still making use of our observations of how you can simplify the radiosity problem.

You should notice that the light levels differ. This is primarily due to the differences in how lights are specified and computed between the two products. It's not easy to set up an identical scene between the two.

However, notice that all the colours, shapes and shadows of the scene are present in both. They aren't identical, but they are both compelling believable images. True physical accuracy is not a requirement for perceivably beautiful output - this is very much our focus.

Perceptual Influence of Approximate Visibility in Indirect Illumination

Insu Yu* Andrew Cox* Min H. Kim* Tobias Ritschel† Thorsten Grosch‡ Carsten Dachsbacher‡ Jan Kautz*
University College London* MPI Informatik† Universität Stuttgart‡

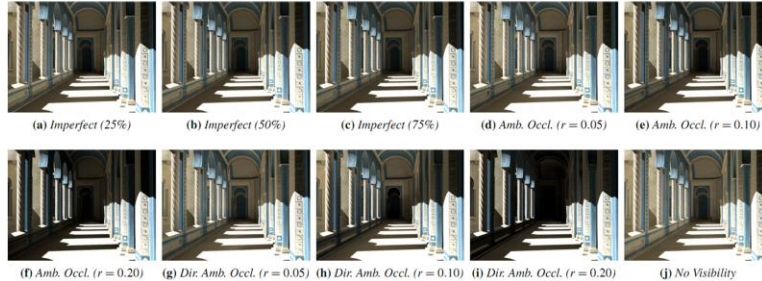


Figure 1: Renderings of the arches scene, where the indirect illumination in each image is computed with a different visibility approximation. Our psychophysical study shows that many of these visibility approximations produce images that are perceptually very similar to reference renderings (cf. Fig. 3).

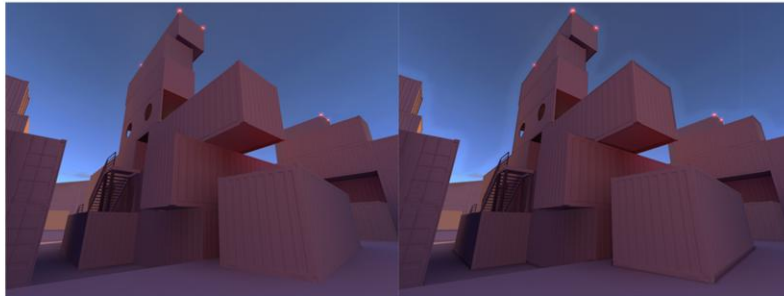
Our observations about accuracy of visibility are not original, but we were surprised by the extent to which you can push it.

This observation was one of several that set the groundwork for Enlighten2.

Insu Yu et al (UCL) unpicked this same observation in a perceptual study. It gives several examples of how you can corrupt the visibility data and how this corruption affects the perceived image quality.

Radiosity and AO

- Early observation:
 - › AO and radiosity play nicely together
- Split lighting integral by distance?
 - › Layer techniques



London Graphics Seminar 2011

Geomerics

Another early observation was that AO and radiosity complemented each other, so various efforts were put into AO techniques.

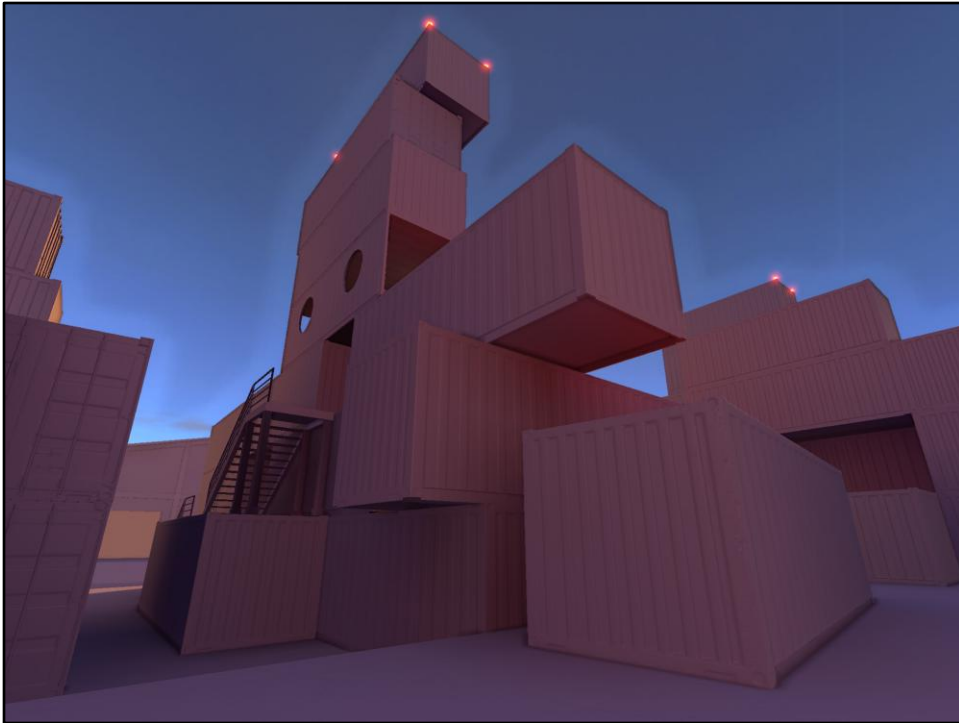
AO, particularly SSAO, can divide people 😊. Personally I think it has a role to help define shape in images, even “if only” for the sake of art direction.

However, there is arguably another more mathematical reason why the two complement each other: They are both approximations to the same thing.

AO techniques that integrate a binary near/far function is modelling light transport with heavy assumptions about the lighting environment. That the near geometry is unlit, and the far geometry is. Conveniently assuming $\text{far}=1$ allows you to modulate in distant lighting afterwards. Assuming near geometry isn't a light source is often a reasonable assumption.

The main reason of discussing this today is to note that this observation points to the possibility of layering several indirect lighting techniques. Where they are split by the range of distances they approximate light transport for.

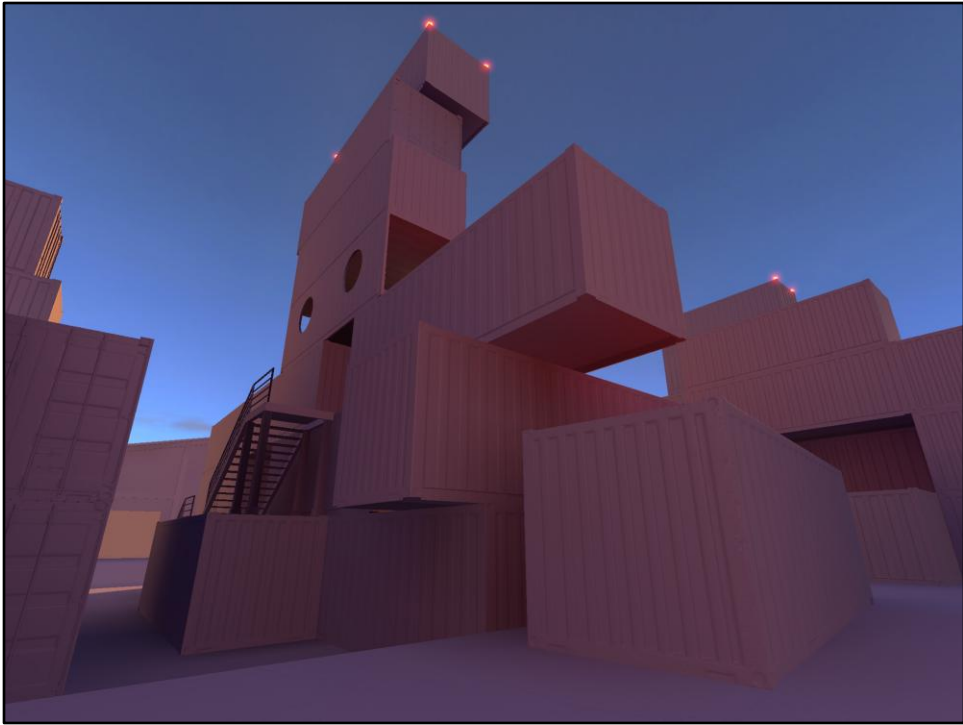
Geomerics haven't yet fully explore this idea yet. Layering AO and indirect lighting works well in enough for our customers at present, but we may return to it.



In case the insert comparison was hard to see:

This is with SSAO.

Screenshot taken from our recent “dockyards” demo in Unreal 3. This is using “detail lighting” viewmode and the typical Unreal 3 AO.



This is without SSAO.

Agenda

- Enlighten
- Research highlights - Past
 - › Observations on accuracy
 - › Lighting research with UCL
- Research highlights - Present
 - › Enlighten CUDA
- Future thoughts
 - › Enlighten as a pipeline stage
 - › Future graphics pipelines with Enlighten
- QA

So, that's a few highlights on past research. I'd now like to turn to our recent work, and some thoughts on the future.

Enlighten CUDA

- **Most recent R&D**
 - › Data parallel CUDA runtime
 - › Shipping in SDK this Friday
- **Extremely fast**
 - › 2-3 ms for complete update of game level
 - › All on GPU
- **Makes use of GPU interop**
 - › All lights using dynamic shadows
 - › Lights with movie overlays

London Graphics Seminar 2011

Geomerics

Enlighten targets the current consoles as it's primary platform. However, there's a increasingly large performance jump between this and high end GPUs and CPUs. So, we have added a CUDA port of our runtime to the Enlighten SDK.

We did originally have Enlighten on the GPU running through a DX9 renderer, but moved away from this fairly early on. The original DX9 path was very restricted by what we could express with pixel shaders. Often forcing us to opt for brute force approaches where we would rather not, and ruling out a whole class of optimisations. We actually got a significant speedup moving and tailoring our tech for the CPU.

CUDA, and 'compute' SIMT code in general is very different from the old DX9 days. We have the additional flexibility we need and the SIMT style approach is a great match. We have been very impressed with the performance levels we can get, so this is looking to be a very promising architecture for us.

As well as speed and a SIMT programming model, we also benefit from having everything on the same die, with the same memory. In particular, we can feed the point sampled input to Enlighten directly from shadow mapped lights, and lights with animated overlays.

So the bottom line is that this architecture is extremely effective at this kind of graphics compute workload, and having all processing on the same side of the bus allows some cool additional features.

Enlighten as a pipeline stage

- Enlighten runtime
 - › Non-traditional stage in the graphics pipeline
 - › Not alone on hardware with unified memory (spus!)
- Data parallel, compute-style code
- Input and output to/from other GPU stages
- Largely asynchronous to rest of pipeline
- User driven, not fixed once-a-frame
- A **compute** pipeline stage

London Graphics Seminar 2011

Geomerics

I think it is worth thinking about how Enlighten fits into the current graphics pipeline.

At present, although it interacts with the gfx pipeline, it's somewhat separated. Even when running on the GPU using CUDA, it has to straddle two separate APIs.

The success of Enlighten CUDA underlined how much resource is available these days, and how much you can do with the additional programming flexibility. Which begs the question, what will we do with all this power, and how do we properly harness it?

In particular, what and how large is the role of compute code in the graphics pipeline?

For instance, is Enlighten in a special case, or are there other similar compute graphics applications? What other graphics can we do in compute stages? I believe research in this area is thin.

Similarly, should compute stages replace any of the existing hardware pipeline stages? For example, do we really need a hardware tessellator, or should a compute stage handle this?

What about hardware rasterisation? There has been considerable interest in research that suggests alternative rasterisation strategies that either require more programmable rasterisation, or new custom hardware. Which route should we choose? How much performance would we need to, and be prepared to, sacrifice?

Programmable gfx pipelines

- **Obstacles**
 - › API for unified compute/fixed-f pipeline does not exist
 - › CUDA and OpenCL graphics interop limited
 - › Switching into compute code is expensive
 - › ...All mostly software issues
- **Ideal:**
 - › User defined DAG of graphics pipeline stages
 - › Mixed CPU/GPU stages on unified architectures
 - › Mixed fixed (e.g. rasterisation) and compute stages

London Graphics Seminar 2011

Geomerics

Beyond the consoles, CUDA is the first serious platform where we get both computation power and compute capability. The PCI-E bus previously killed this on PC. While our experiences with CUDA are very promising, it also highlighted some fairly major software issues.

The division between graphics and compute is noticeable. There are omissions such as texture arrays (now in CUDA 4) and cube maps, and the inability to use hardware to read compressed textures. It's not possible to drive much of the hardware from compute.

It is also very expensive to switch between CUDA and graphics modes, so your frame typically breaks into a compute and rendering section. Fine grained switching is not practical.

And on a more general note, having to work with two APIs does have its own overheads. And CUDA is not trivial to code. So the programming environment, whilst powerful, is still a significant development burden. Certainly a considerable way from a programming model I would feel content with.

Future directions

- Many advances in rendering coming out of research
 - › Stochastic rasterisation
 - › Micro-polygon rasterisation
 - › Decoupled sampling
 - › ...All require increased programmability and/or adapted hardware
- Many open questions
 - › How to mix fixed function and compute
 - › Role of code generation in compute code
 - › What other gfx advances can be made through compute stages?

London Graphics Seminar 2011

Geomerics

So this is a new field of possibilities. Unified architectures will become the norm. Software is lagging behind hardware, but further graphics and programming model research can help define the APIs and direction for the future.

A large question for me - that even Larrabee's lack of graphics success has failed to put to bed - is whether we really need all the custom graphics hardware we currently have.

Yes, we have proved that custom hardware and hardware-tailored algorithms are required to achieve the performance levels we see on our current graphics workloads.

But what if we did things differently? As one example, what if we replace the current rasteriser with a compute-based one, but gained visual quality from the additional programmability? This is not a like for like comparison.

Or perhaps we accept a performance hit, but gain from a fully software pipeline that allows us to develop software more quickly and flexibility. How would this change how we develop games today? The cost of development is our most pressing concern. How much value do you place on making your graphics unique? How much performance would you sacrifice?

None of these are simple questions to answer, and I hope you have some ideas!

Thank you! Questions?

- Demo!
 - › “Dockyards”
 - › Fully dynamic UE3 level running on consoles
- Email your after thoughts:
 - › sam.martin@geomerics.com
- Licensing:
 - › sales@geomerics.com



London Graphics Seminar 2011

Geomerics

I'd love to hear your thoughts and feedback. Email me!

Thanks very much. Questions?