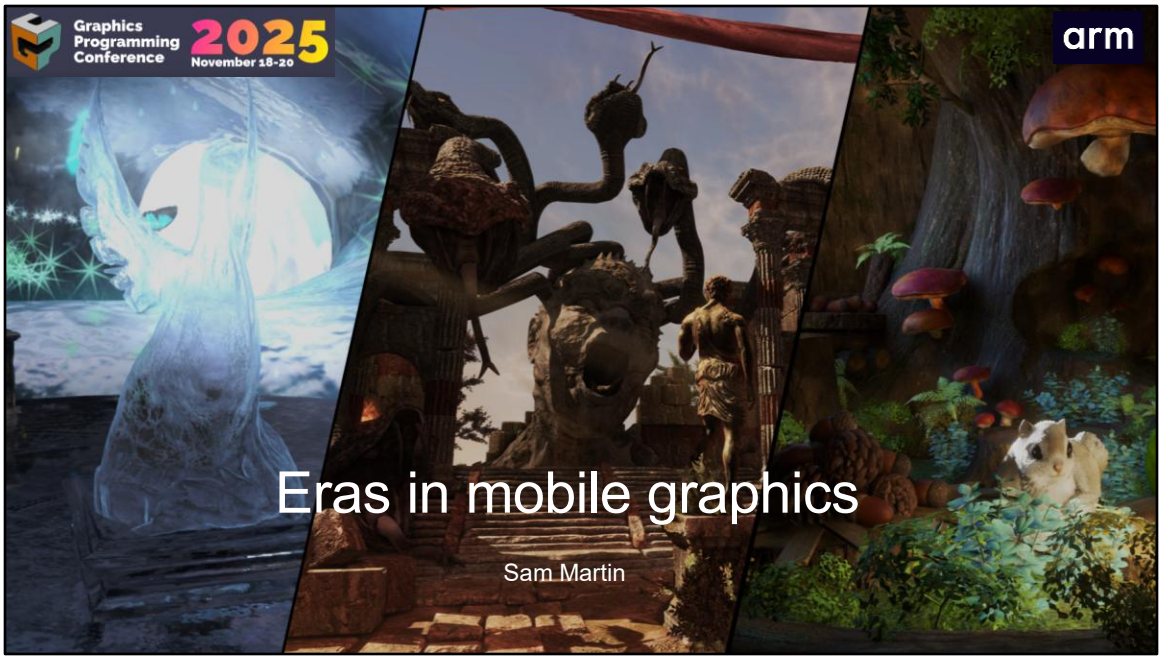
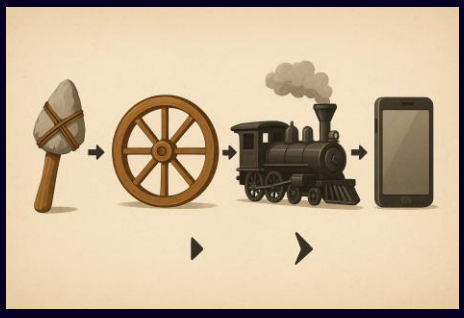

Graphics
Programming
Conference
2025
November 18-20
arm
Eras in mobile graphics
Sam Martin

This talk

- "Eras" as an excuse
  - Range of topics
  - Across a range of time
- Not a super technical talk
- Bold claims, a lot of squinting / extrapolating
- Encourage thinking / talking / complaining
- Kick off the conference!

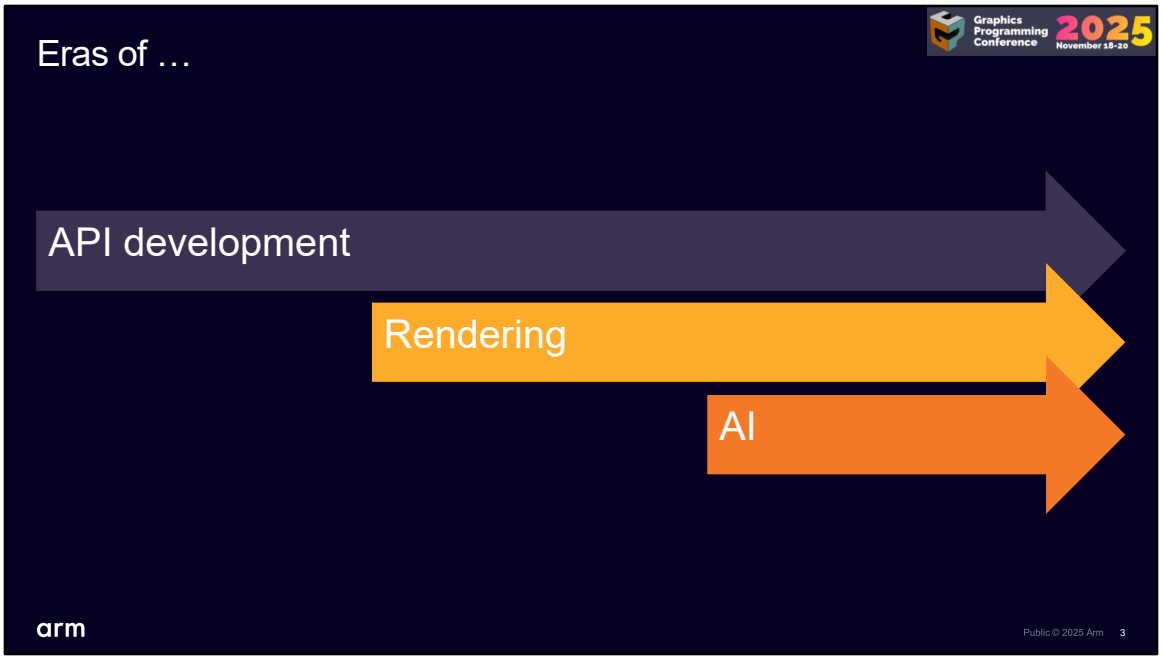


ChatGPT made illustrations like this one

"Eras" as an excuse
- Range of topics
- Across a range of time

Partly for fun, partly to see what lessons we can learn

Use this to observe what the new era we are entering will look like
- Spoiler – this era will involve AI



Something historical

Something current

Something emerging

A bit about my background (so you know my biases)

Graphics Programming Conference 2025 November 18-20

- Ex-Game developer (PC, PS2/3, Xbox360 era)
  - Black & White 2
- Ex-Geomerics
  - Enlighten
- 12 years at Arm working on technology for mobile graphics
  - On chip rendering techniques
  - Various VR/AR technologies
  - Future content trends and optimisations
  - ~5 years now on neural graphics / ML

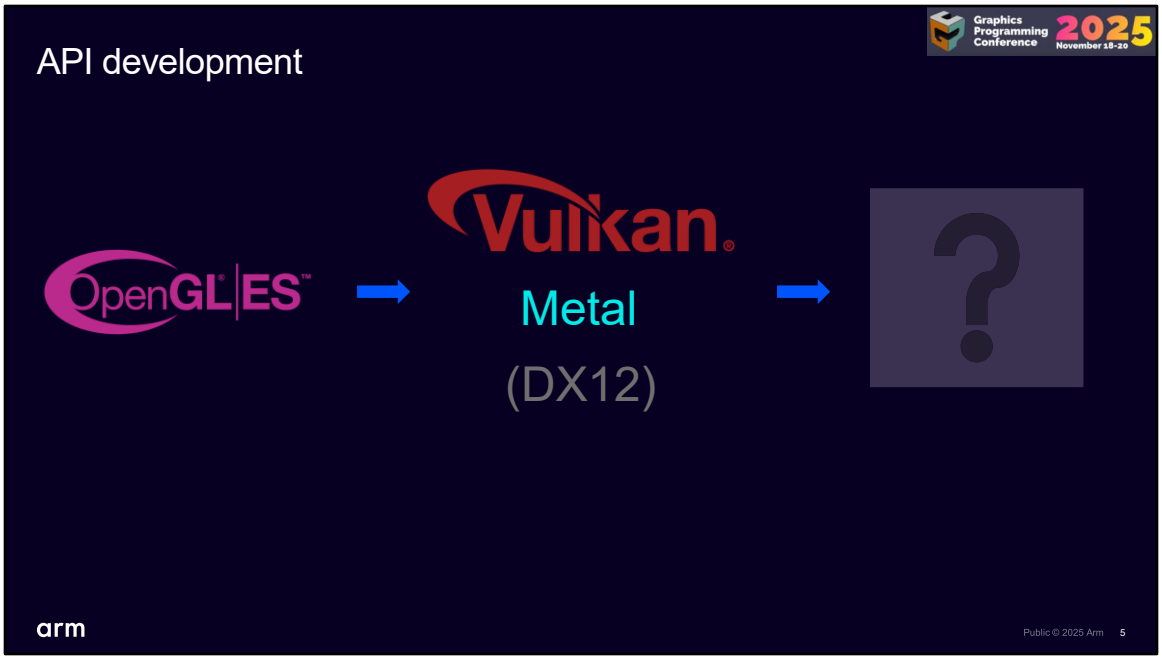
- Biases:
- I work on Arm's Mali / Immortalis GPU line – focus on Android-Vulkan
- I lead a GPU/ML technology project in Arm, bringing AI acceleration to GPUs

arm

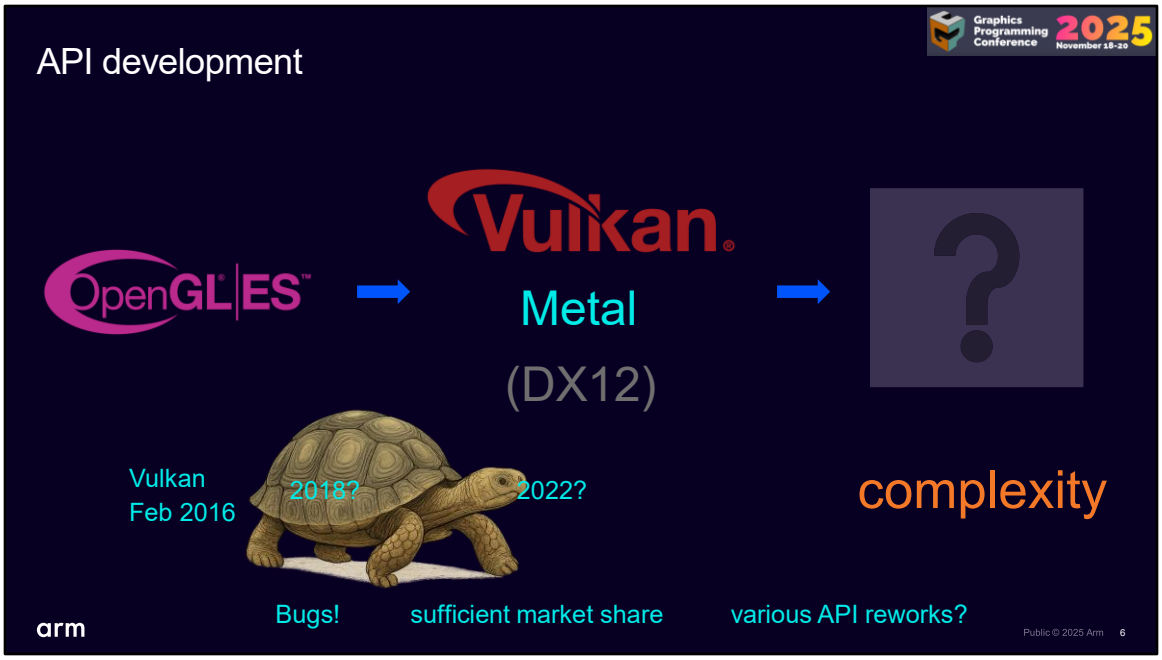
Working in mobile graphics for 12 years at arm, and before that at Geomerics.

I used to make console and PC games, then went on to make games lighting technology, notably Enlighten. I've not actually made a mobile game as I was deep into the technology side by that point. I have however been involved in exploring most cool new technologies that came to mobile.

When I joined Arm they were in the process of transitioning to their first scalar architecture from a VLIW architecture. At that point, most content was a single renderpass and compute shaders were a new thing.



First lets start with eras of APIs



Getting to “Vulkan by default” took a very **very** long time.

Vulkan launched Feb 2016, Metal 2014, DirectX12 2015

All required new HW

Remember - Vulkan “delivered”!

Low overhead, explicit control, predictable perf, matched to HW

But..

Bugs: initial HW being viable – maybe 2 year delay + reputation damage

Reaching sufficient market share – but that only excuses us for, say, another 4 years?

Takes us to 2022 – 3-4 years ago

Clear then that something had got a bit stuck

Descriptor model and sync all had heavy revisits – more reputation damage?

World of “device lost” pain

Complexity remains a major thorn in the side

- LoC to first triangle is huge, gargantuan spec document, painful when things go wrong

Can we clean it up?

- Very tempting but has to be via additions
- Must accept as sunk everything pre-existing

Can we wrap it in something simpler?

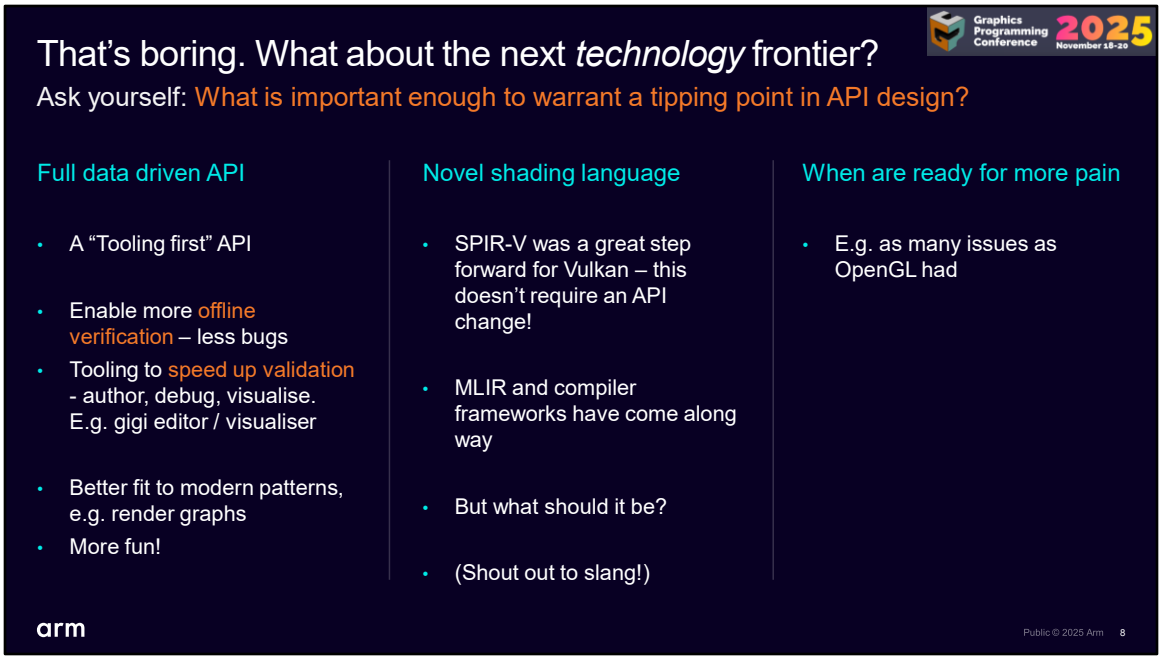
- Make simple things simple again!
- Provide a simpler Vulkan-like layer on top of Vulkan?
- Possible, but non-trivial investment and who would it target?

Maybe it's already happened: WebGPU

- A simpler modern API, building on and unifying existing APIs
- F5 = compile + deploy, is a killer feature
- Web technology isn't really about web pages anymore

I'd observe that Vulkan is like the C++ of graphics. It's a tool with which you can do whatever you need to, but not always without some manual pain, and has old

features you want to steer away from that didn’t stand the test of time



We cannot change what has already shipped, and WebGPU may address the need for a simpler API.

So what does that leave?

There is still a case for make bleeding edge development more efficient / effective
"make graphics easy again" – address an imbalance of effort to reward
Not simplification → streamlining, increasing utility/robustness

Possible visions:

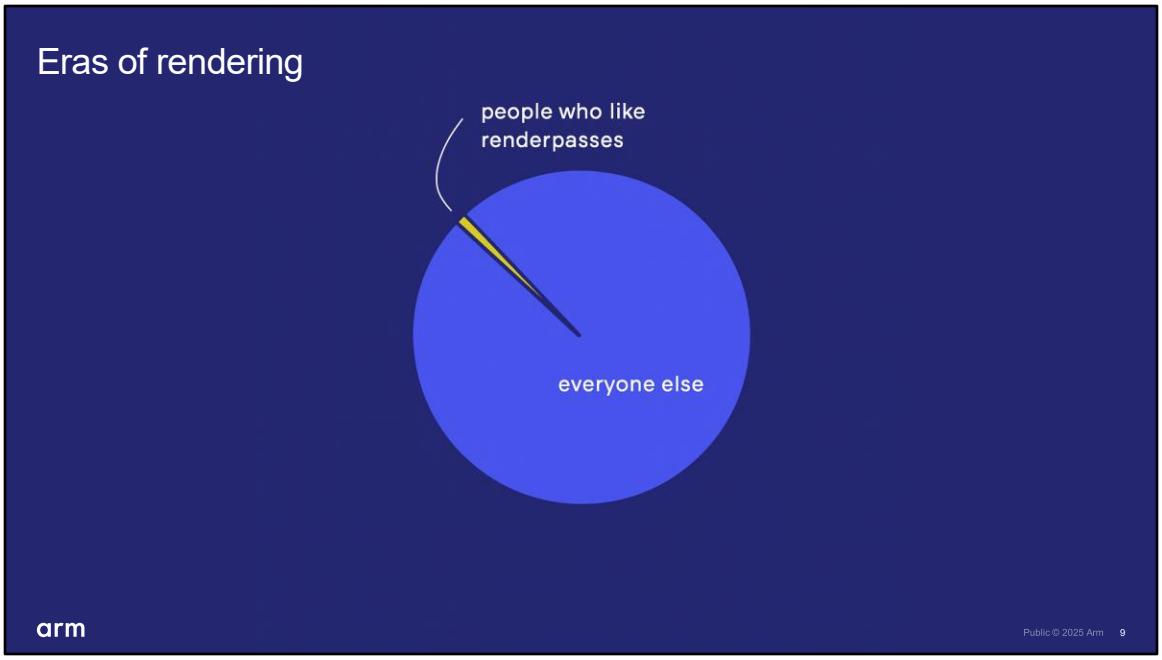
A tooling-first API – a data-driven / IR-first / "No API" API
Enable verification - mechanical, offline functional verification of correctness.
Do more offline, less runtime debugging
Speed up validation - tooling to author, debug, visualise data flows. E.g. gigi editor / visualiser
More natural fit to modern patterns, e.g. render graphs
Make graphics easier / more fun

Revisit how shaders are authored

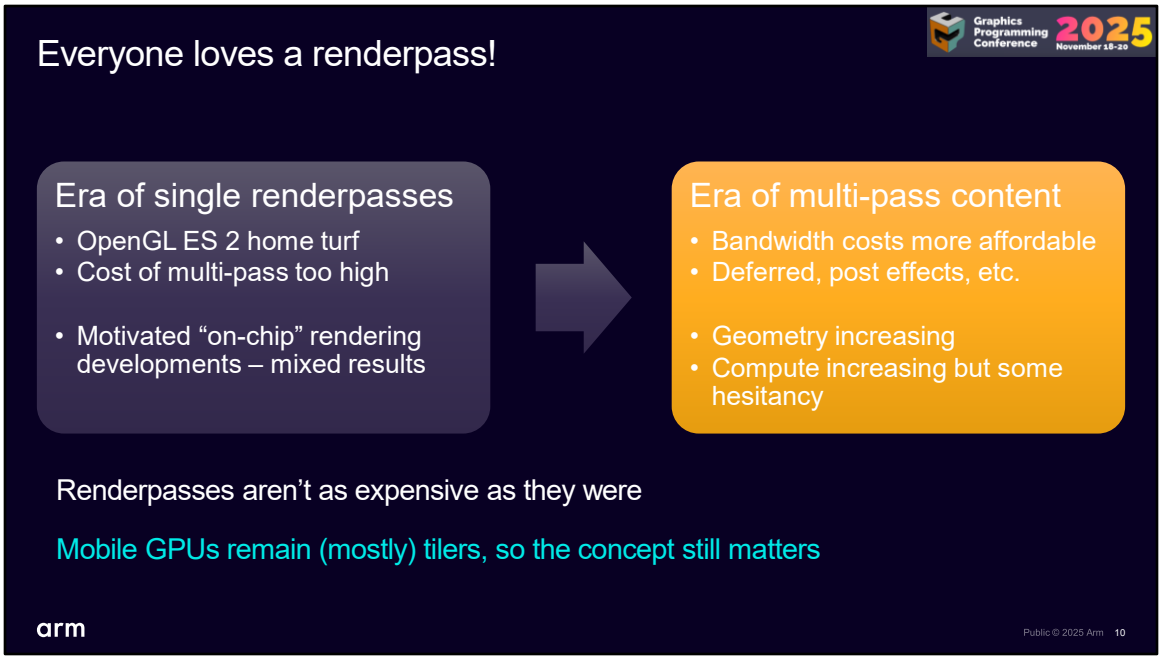
- Noodle-graphs have limits, text-based representations scale to greater complexity
- Move to C++ or Rust?
- Shout out to slang!

Challenges to consider:

- GPU-driven – how on earth do we tool and debug that?
- How to best harness AI



Maybe we can learn some other lessons from the past though.



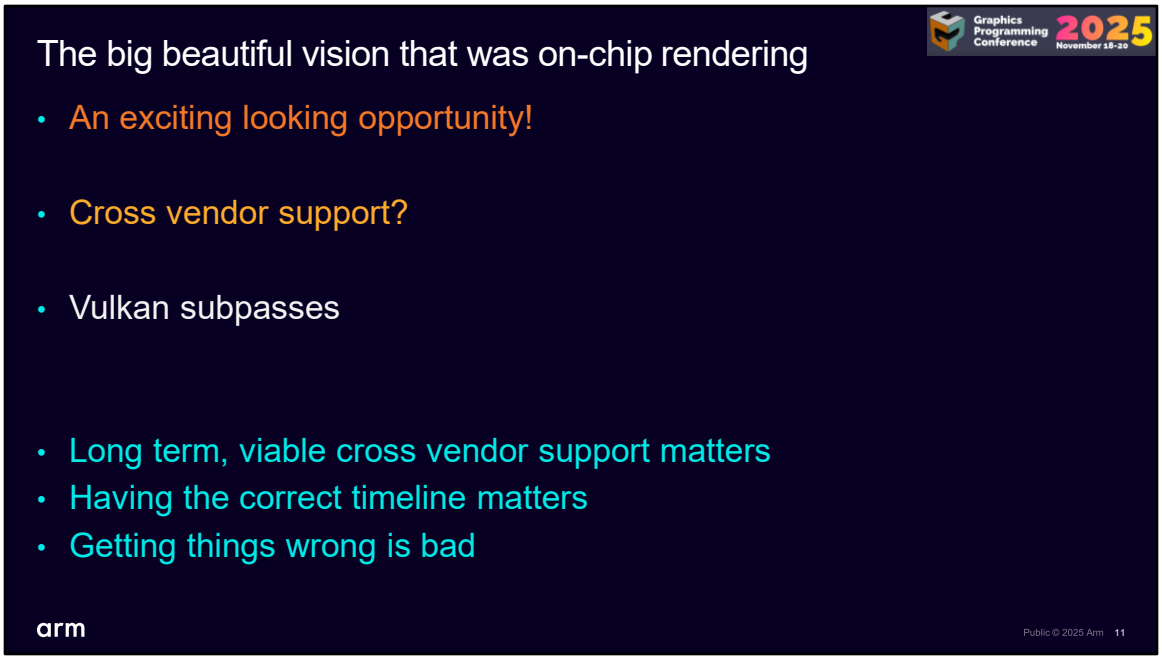
First, there was an era of (mostly) single forward+msaa renderpass content

- OpenGL ES 2 home turf – although renderpasses not actually visible in the API!
- Cost of moving to multiple passes (e.g. deferred) was non-trivial and became a barrier
- Motivated “on-chip” technology developments – with mixed results

Then, we move to an era of multi-pass content

- Bandwidth costs now more affordable – deferred rendering, post effects, etc. now common
- Also seeing increasing geometry costs
- Significant hesitancy to adopt actual compute shaders due to associated HW costs (and lack of need?)
- Still a huge amount of fragment-based content today

Renderpasses exist because they remain an important concept for tilers, but they aren’t as super expensive as they used to be.



There's perhaps some lessons to learn from the era of "on chip rendering".

At the start I was working on Enlighten, which we had running on mobile. Dynamic lighting was a major cost. Not much point having dynamic radiosity if you can't render the direct lights! BW of deferred was a huge problem – 4-6 GB/s. Each GB/s was ~100mW. We started speaking to HW vendors about this. I'm not sure where exactly the idea came from, but there was something of a collective recognition that the tilebuffer in mobile GPUs could offer per-pixel read-write access (not just write access). That's a big deal for lighting because it allows you to render your gbuffers, then render your light geometry and read back depth normal to compute lighting, all in a single pass. It can be a significant bandwidth win.

Arm were pioneers of this and there were several rounds of experimentation and API designs. Pixel local storage emerged for GLES but adoption was limited by cross vendor support. We learned late that others couldn't support it. In hindsight it would have been better to deliver less but deliver it more broadly (e.g. framebuffer fetch), but we didn't know this at the time.

Vulkan then came into being and with it came subpasses - undoubtedly the wrong API, which emerged from the frantic all-hands-on-deck exercise of shipping Vulkan 1.0. There was just insufficient time to get the right API.

By now, we had been working on this a long time. We were further from the original more explicit ideas in the early APIs, but also weren't as minimal and cross-vendor as frame buffer fetch. We had to introduce a "did it merge?" API to provide feedback to developers. Significant parts of the API were of no use to any vendor, and subpasses a cost to all developers.

Sadly, this kind of sealed the fate of on-chip rendering in Vulkan, in my eyes at least, although things have since improved with dynamic rendering local read and Apple have tile shaders. There is still value in these techniques. AR/VR devices stand out as an interesting example. But for wider mobile gaming it is a technology who's moment of peak impact has passed. We will shortly see devices with peak 100+ GB/s, and we are looking instead at different ways to do dynamic lighting.

I learned a lot from this experience. It taught me a lot about the vital importance of building and getting cross industry consensus for technology. Longer term this is essential for a technology to succeed. The role API design has in that is huge. It also taught me a lot about the timeline involved, and the costs of getting things wrong.

Future of tiling?

- Subpasses have been killed, renderpasses live on
- Moar meshes! → moar tris! → moar tris/pixel! → bad news for tilers?
- Should tile-based rendering also die?
- No
  - Not yet justified
  - Important to scale down
  - Tiling has also adapted
- More importantly, there are bigger shifts on the horizon

arm

Subpasses were then killed off. But geometry continues to increase. Should we kill off tilers as well?

No!

Maybe rasterising triangles will die?

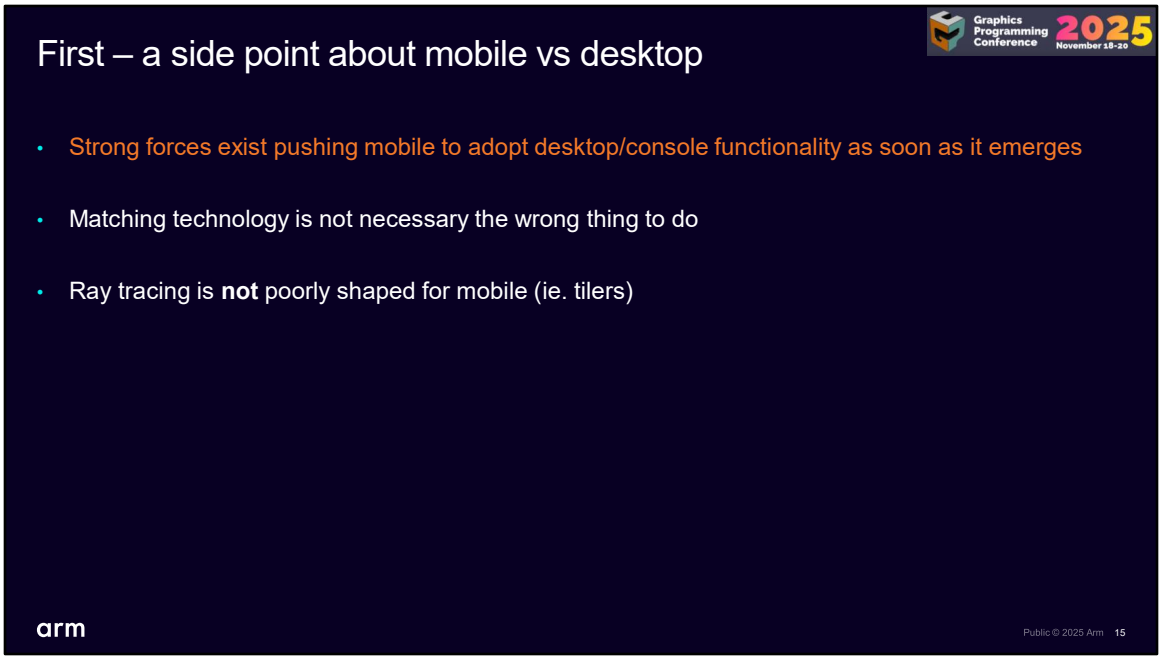
- Ray tracing?
- SW rasterization?
- Gaussian splats?

The everlasting joke about ray tracing is that it was always 5-10 years away, and then when you got 5-10 years into the future, you found it was still 5-10 years away.

Well, now the HW exists.

Adoption isn't quite here though. It's growing bit by bit each year, a bit like Vulkan's long dark night of semi-adoption. But it is growing, and people are exploring.

The light is close to the end of the tunnel, and we think we can see the inflection points for it.



Strong forces exist pushing mobile to adopt desktop/console functionality as soon as it emerges

(The reverse doesn't happen so regularly)

Sometimes this pushes technology onto mobile even if the tech is not entirely ready or suitable

E.g. geometry shaders, tessellation, etc.

If we really want them to run well on mobile, they should be re-designed (not going to happen!)

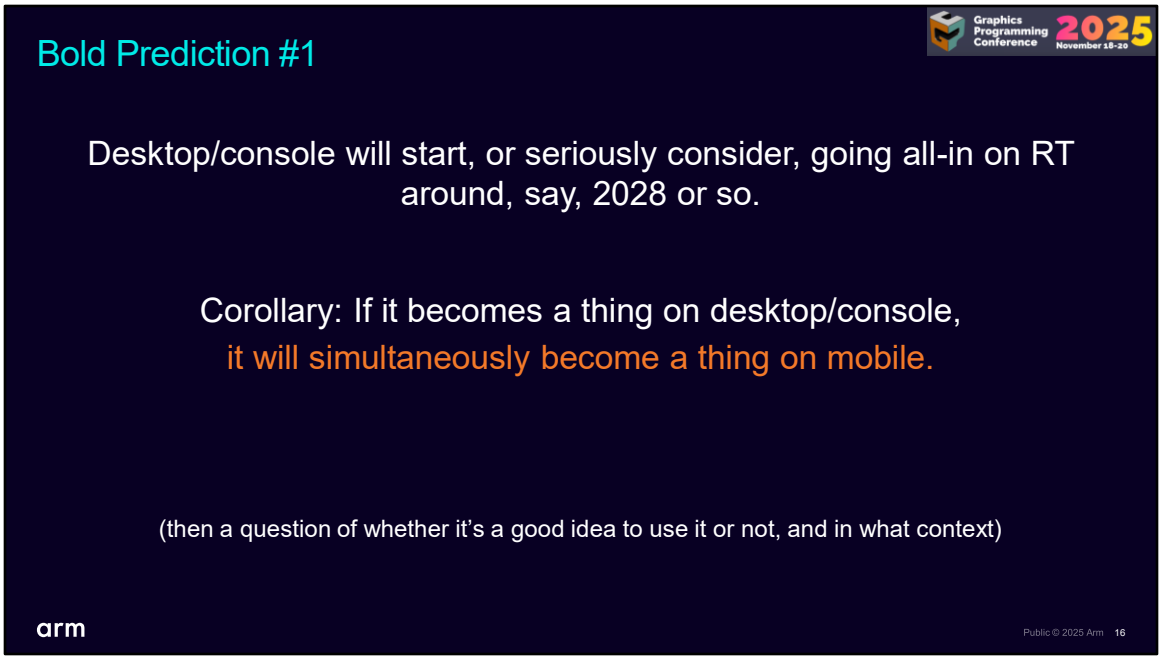
Matching technology is not necessary the wrong thing to do – genuinely useful to many people to have a common technology platform between desktop and mobile.

Code "just works", optimisation a secondary problem

Underlines the importance of introducing technology and APIs that work across both though

But ray tracing is **not** poorly shaped for mobile

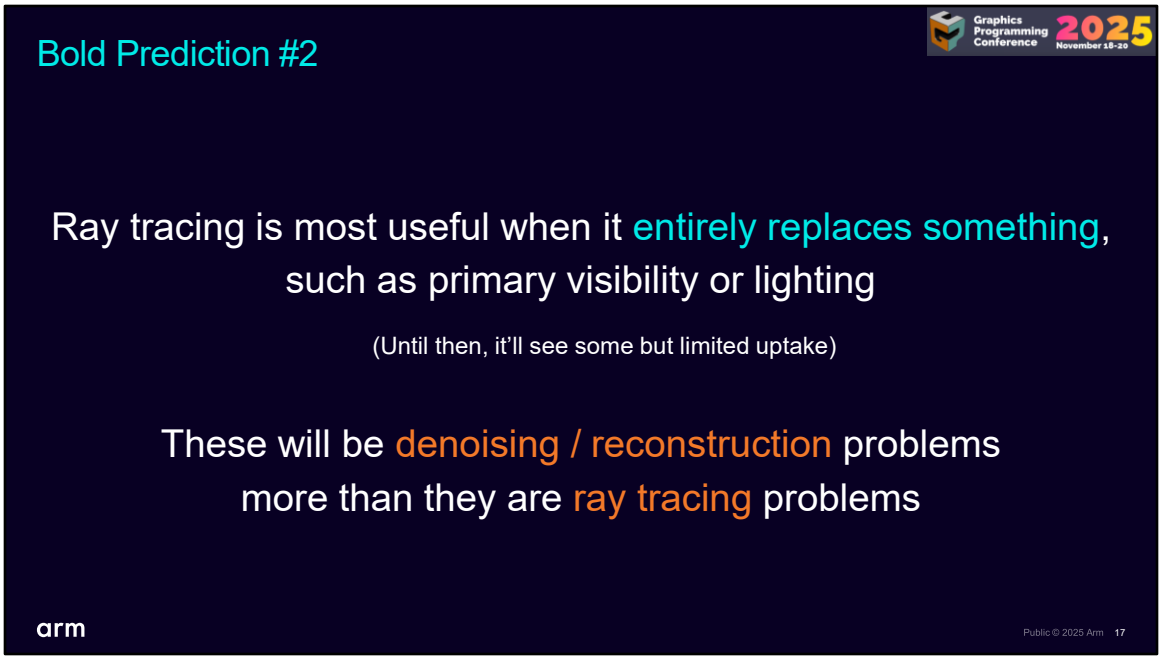
- Yes, it's expensive. Yes, it's expensive in terms of bandwidth
- It's not inherently inefficient or inappropriate for mobile though (unlike, say tessellation)
- Modern mobile HW is actually really good at ray tracing and can/will get better



Desktop/console will start, or seriously consider, going all-in on RT around, say, 2028 or so

- Plausible timeline for new consoles
- Pain in the arse doing a bit of both. API friction, double data, etc. HW becoming more capable.
- At some point, it's just worth RT'ing everything and having one set of problems rather than two.
- Finally get away from the limitations of rasterization

(I was unaware of it at the time I gave this talk but subsequently discovered that Mark Cerny of Sony has spoken publicly about their investment in RT and AI for PS6 a "couple of years" away.)



Ray tracing is useful when it entirely replaces something, such as primary visibility or lighting

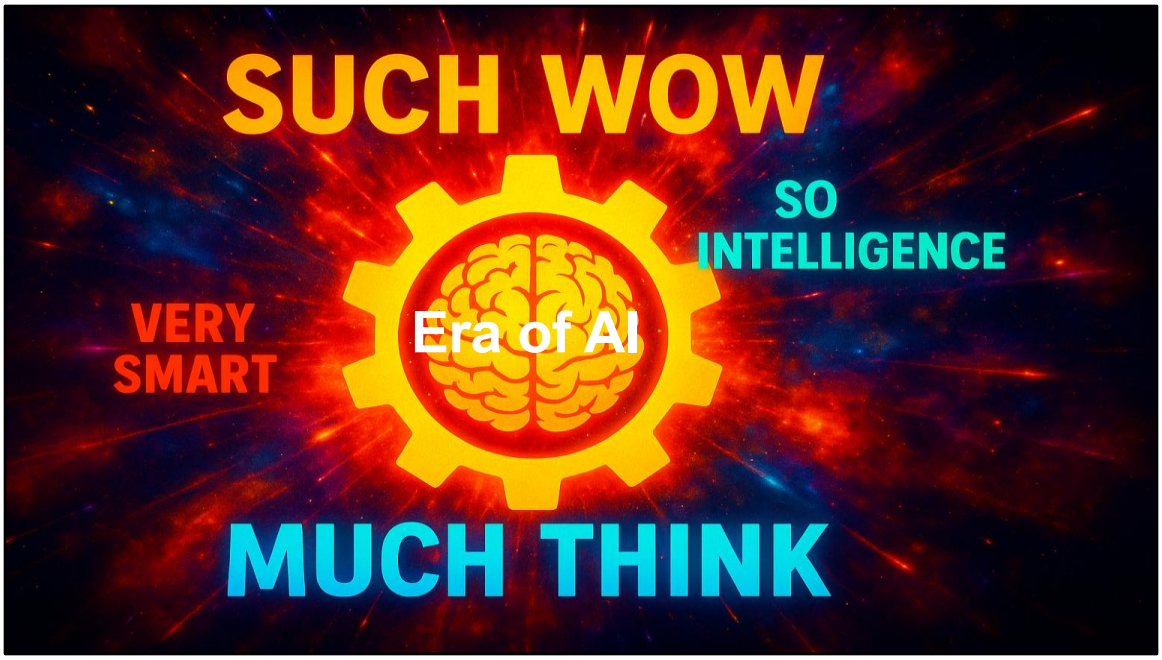
- Until then, it'll see some but limited uptake

Important problems will be denoising problems more than they are tracing problem

- Tracing 1-2 rays / pixel for lighting is becoming affordable on top end devices
- Tracing tens of thousands of rays for some indirect probes / etc – should be fine

Denoising is the challenge

- SVGF-based solutions are expensive
- Quality isn't that great either



And that brings us neatly to…

Neural Graphics at SIGGRAPH in Keywords

Graphics Programming Conference 2025 November 18-20

2018

2022

arm

 19

AI / ML / neural graphics is not a small thing.

Obviously I would say that – I lead a project doing exactly this. But I it's objectively clear that AI is having a major impact on graphics and has been for many years.

This is a moment where new technology up-ends everything we are used to, bringing disruption and opportunity.

What are GPUs for anyway?

- They draw the pictures!
- They are the coolest accelerator!
- …
- They are the **ubiquitous deferred bulk compute accelerator**
- AI will shape GPUs
- What can AI do for graphics?

arm

Unless you are on console, GPUs in most consumer devices are not only for graphics.

In fact, the priority for many GPU vendors is not always graphics.

They have to do the graphics.

But they are also the best means of accelerating many workloads.

They are a ubiquitous accelerator. Sure you can do better if you target something per device, but if you want to write and release code that runs everywhere, and it needs to run faster than the CPU, then it runs on the GPU.

It’s worth noting this because the world of AI is much bigger than graphics.

There are more people working on, it has more investment.

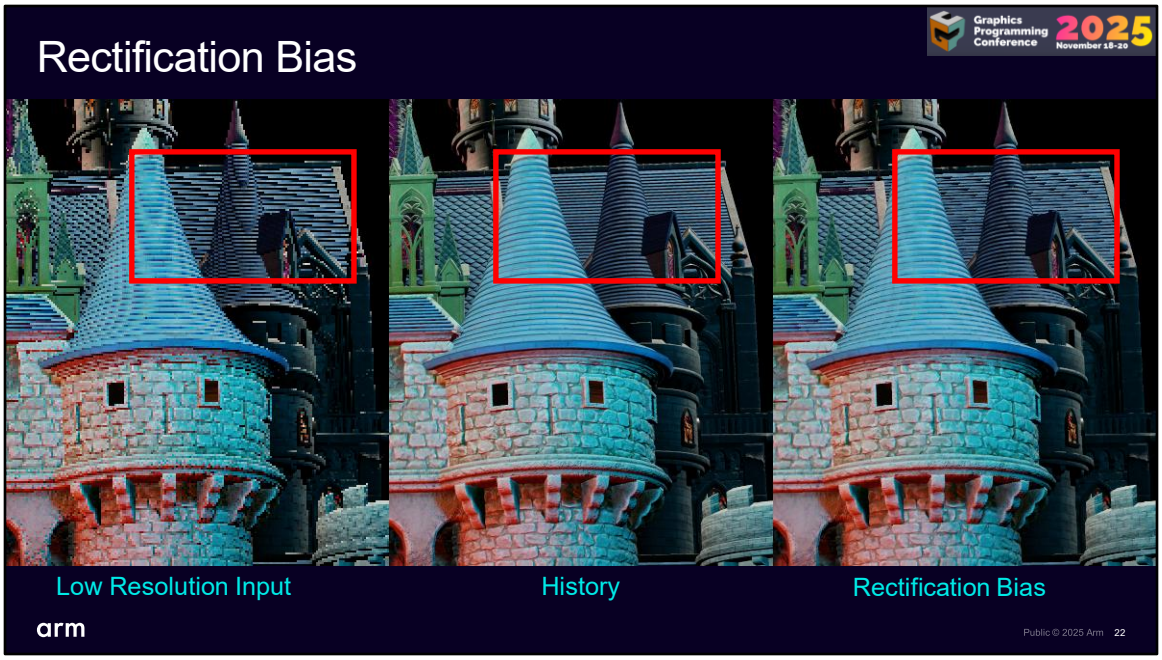
It will shape the future of GPUs.

And we should be looking to what the world of AI can offer to graphics.

Let’s start with what AI can do for graphics.

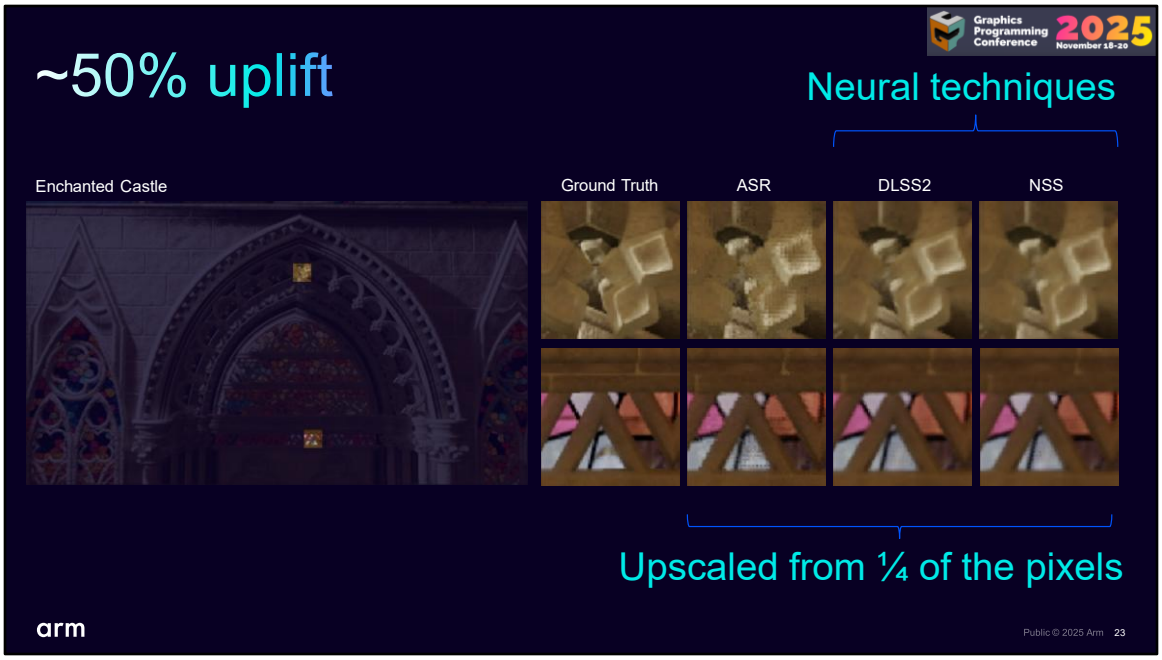
I imagine this kind of image will be familiar to lots of you.

It’s what happens if your heuristics on when to reusing your history buffer get a bit over eager.



But if you reset the history too aggressively, you can induce bias towards the current sample, re-introducing aliasing (see the roof on the right)

This can be seen when applying an AABB clamp on a nicely accumulated history, with a very aliased input, aliasing returns to the output frame.

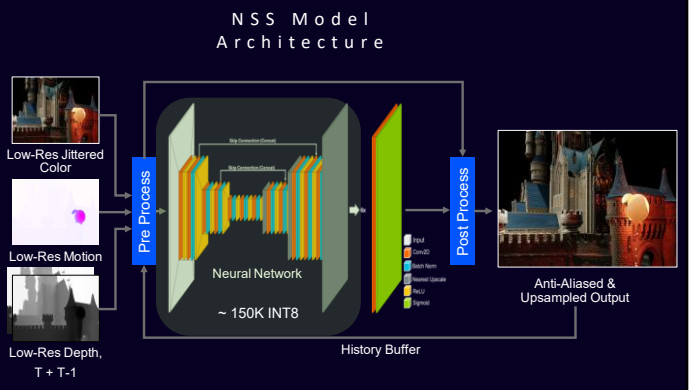


This kind of heuristic-based reconstruction is an area where ML really excels

NSS-like algorithms are going to have a huge impact. These kind of performance uplifts are just not on the cards from any other HW technology in the foreseeable future.

Disruption of Real Time Graphics – The **Conservative** View

- Neural Super Sampling and related neural graphics techniques will transform graphics efficiency
- ML processing of “full scale” images will become a major feature of most, if not all, image processing applications



arm

A lot of the discussion around ML is on the impact of it in non-realtime settings. But what about ML in real time graphics – aka “neural graphics”?

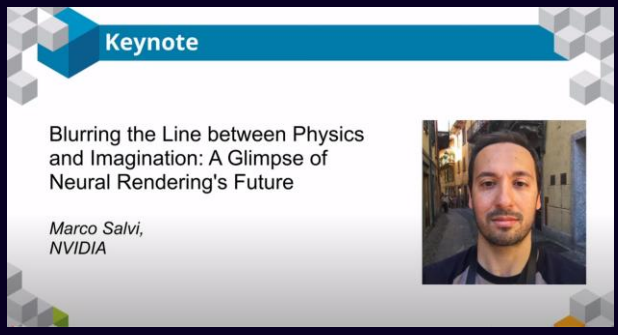
Lets look at this by bounding the future between a conservative and bold view.

We can be certain about the conservative lower bound. We can see this happening today in technologies like NVIDIA’s DLSS. This alone is transformative for real time graphics and GPUs as we know them and demands changes.

Disruption of Real Time Graphics – The **Bold** View

Graphics Programming Conference 2025 November 18-20

- Significant aspects of rendering will be entirely displaced by ML alternatives (e.g. triangles?)
- ML processing will be the technology that enables entirely novel applications



I3D 2023 keynote talk by Marco Salvi

arm

But the bold vision is alive and well at places like SIGGRAPH which is comfortable looking further out.

Here we can think about what else we are likely to see change. Will whole parts of the graphics pipeline see change or displacement due to ML, as Marco Salvi suggested a few years ago?

In graphics more generally, it's perhaps not so bold to suggest that ML will be the technology underpinning future applications

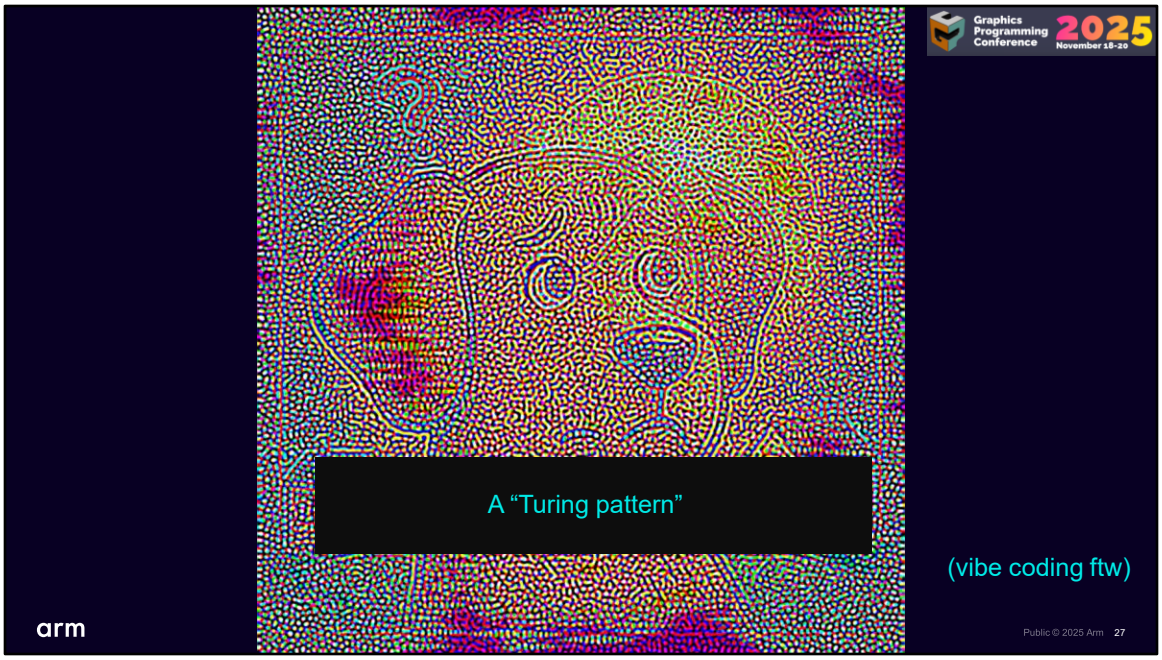
But first a bit of a side quest.

A good number of years ago, I think Alex Evans noted (maybe in a SIGGRAPH talk?) that making things look good was often largely down to "adding noise and blur". This stuck with me. Years on I'd suggest one tweak: Adding noise, blurring **and then sharpening**, is the fundamental chain that makes things look good.

I know it's tempting to go overboard with the sharpening / enhancement stage, but try not too ☺

So stacks of noise-blur-sharpen has a remarkable tendency to good to our eyes.

Why?



I don’t know but I might offer a wild hypothesis:

What happens if we repeatedly blur and sharpen an image?

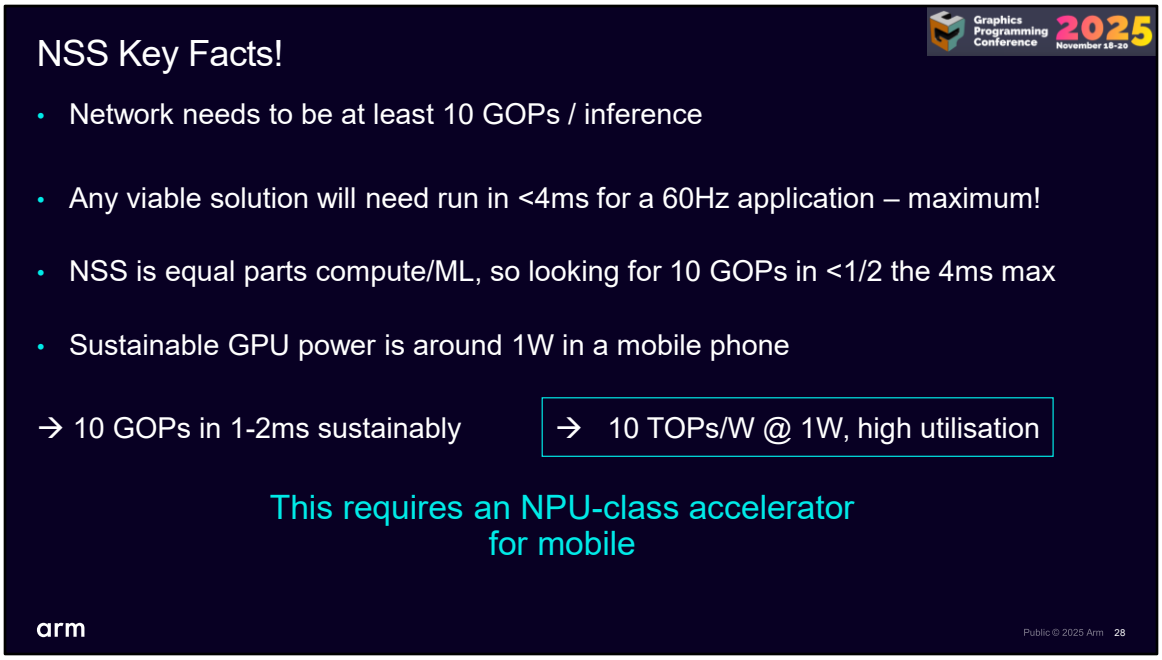
Turns out we get a pattern Alan Turing first noticed, so they are known as Turing patterns.

Turing patterns are observable in biological processes, which turns out to be how Turing found them.

Biology seems to like filtering of random, diffused processes and it’s something we therefore unwittingly observe all the time.

So maybe that’s the reason!

Anyway, blur+noise+sharpen is still a thing today, but now we also have “stochastic everything + reconstruct”. That’s what underpins NSS and it so happens that this is an area where ML really excels



So what do we need to build ourselves an NSS?

Now we know it's possible, so here's the key figures that come out of our NSS work (all published)

Our work showed that to get good results you need a network with at least 10 GOPs/inference (quantized int8). With ML, bigger is better, so this is the baseline.

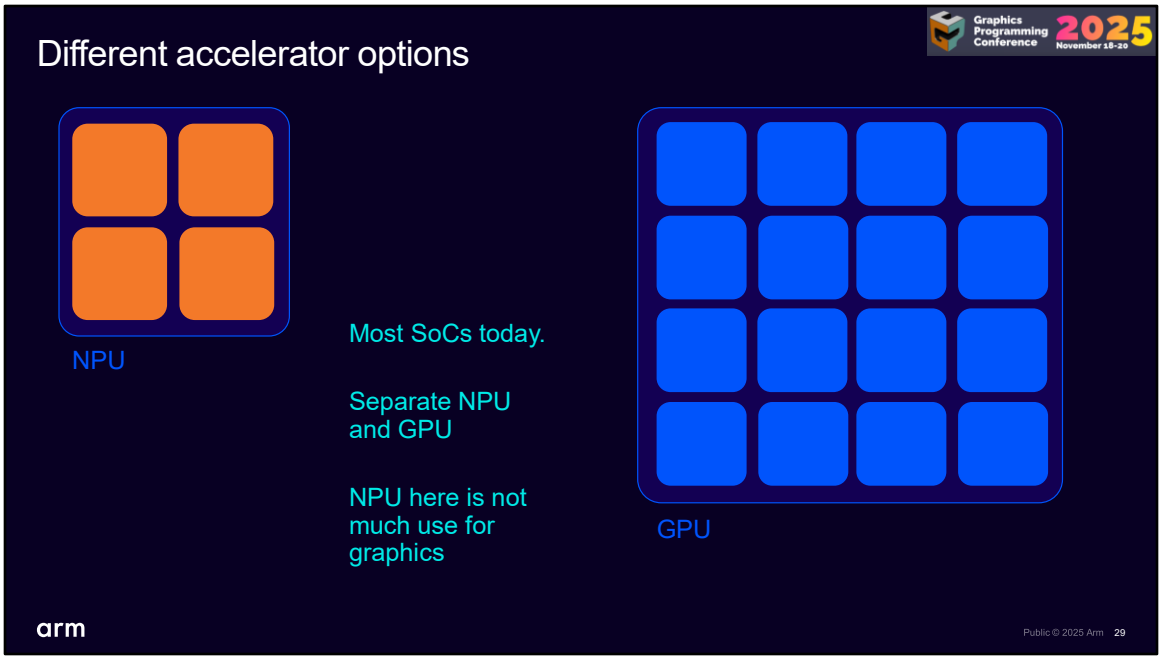
We can also observe that if we are looking to see more than a 25% FPS uplift in a 60Hz app, then we cannot take longer than 4ms.

Why 4ms? Because if you get a 2x speed up on a 14ms frame with a 4ms solution you are ~25% up (11 vs 14 ms). Anything slower than that and it's probably not worth it

Our network is also roughly half-and-half compute shaders and ML network, so we get a target of 10 GOPs in 1-2ms

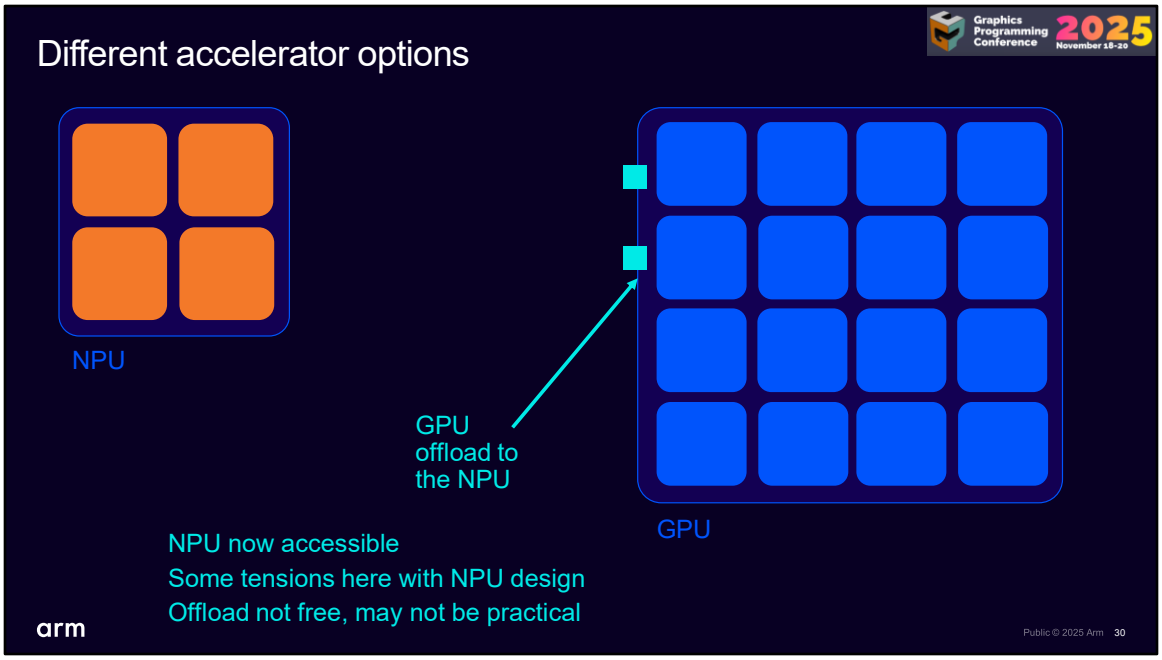
To do this in a sustainable 1W mobile budget, you therefore need good utilization on a 10 TOPs/W accelerator.

In essence, **this means we need an NPU-class accelerator in mobile.**



So what are the different options for doing that?

Well, there's what we have today, which is separate NPUs and GPUs. These separate NPUs are not much use for graphics – they have different APIs, the NPU doesn't even have a public standard API, they are largely intended for system use. It's just not set up for this kind of use case.

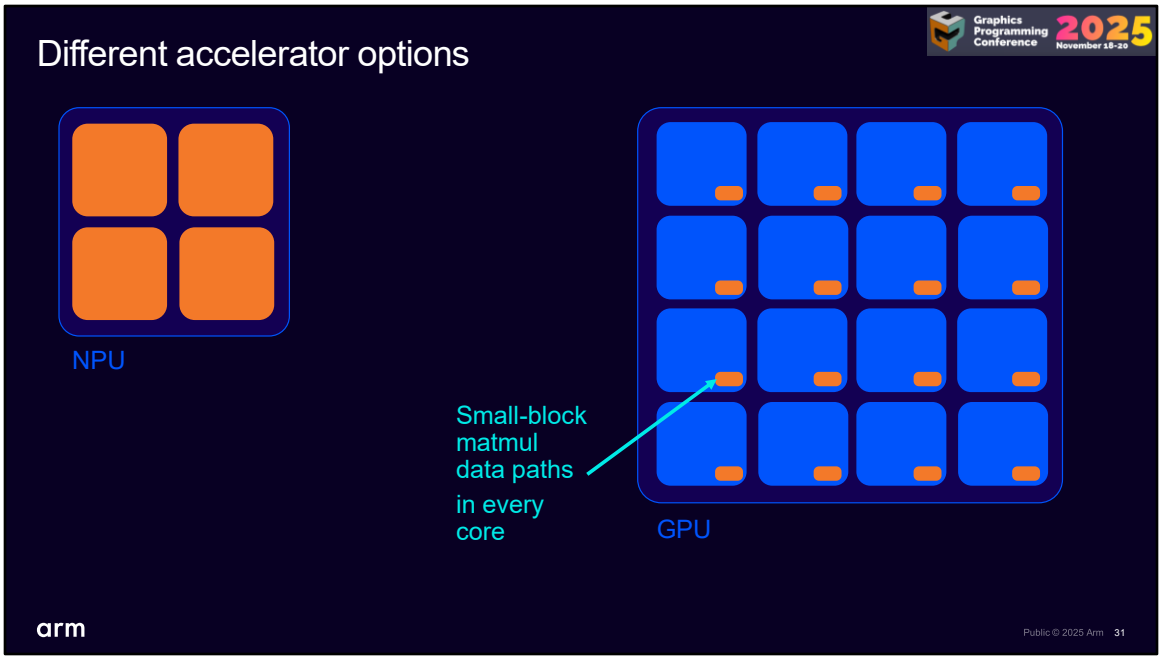


We could try joining these things up?

Apple shipped something like this recently in Metal 4

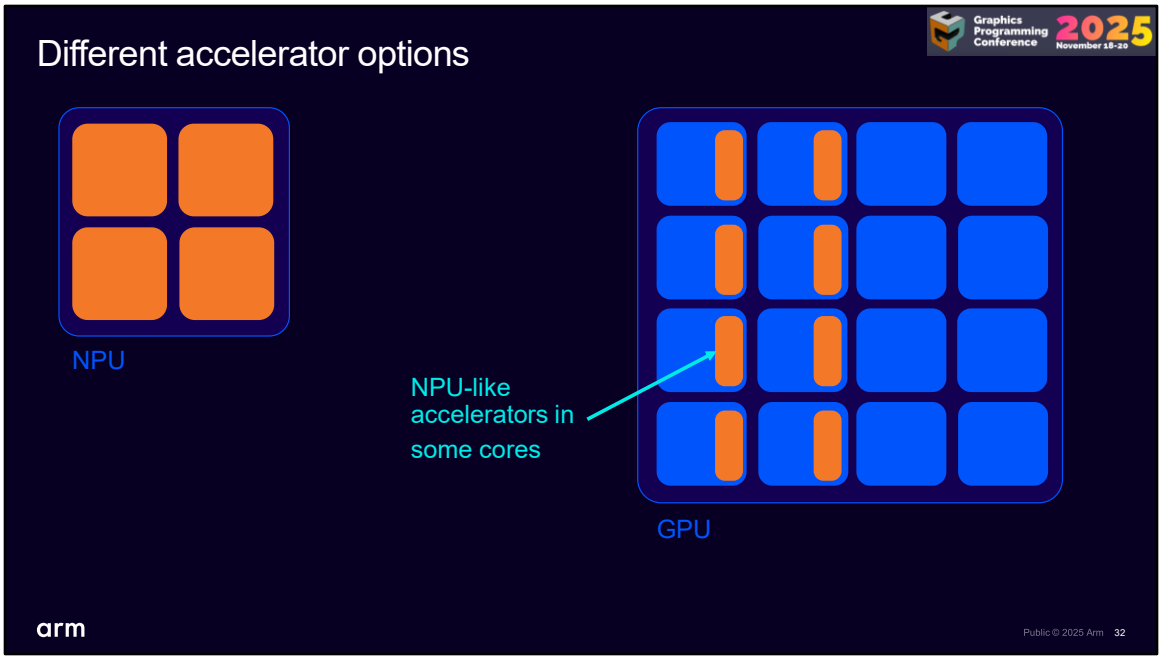
You can compile work for the NPU offline, package it up and then dispatch it from the GPU. No runtime API to change things – just bind and dispatch.

However, there is tension here. Apple have complete ownership of their stack, HW and SW. Other vendors will have more challenges. The offload and sync is not free so there is a minimum size offload. It also requires alignment over fiddly details like how to marshal data. In many cases, the NPU and GPU are written by different teams in different companies, or are working to different strategies.



Another option is to extend the existing GPU with a small block matmul data path.

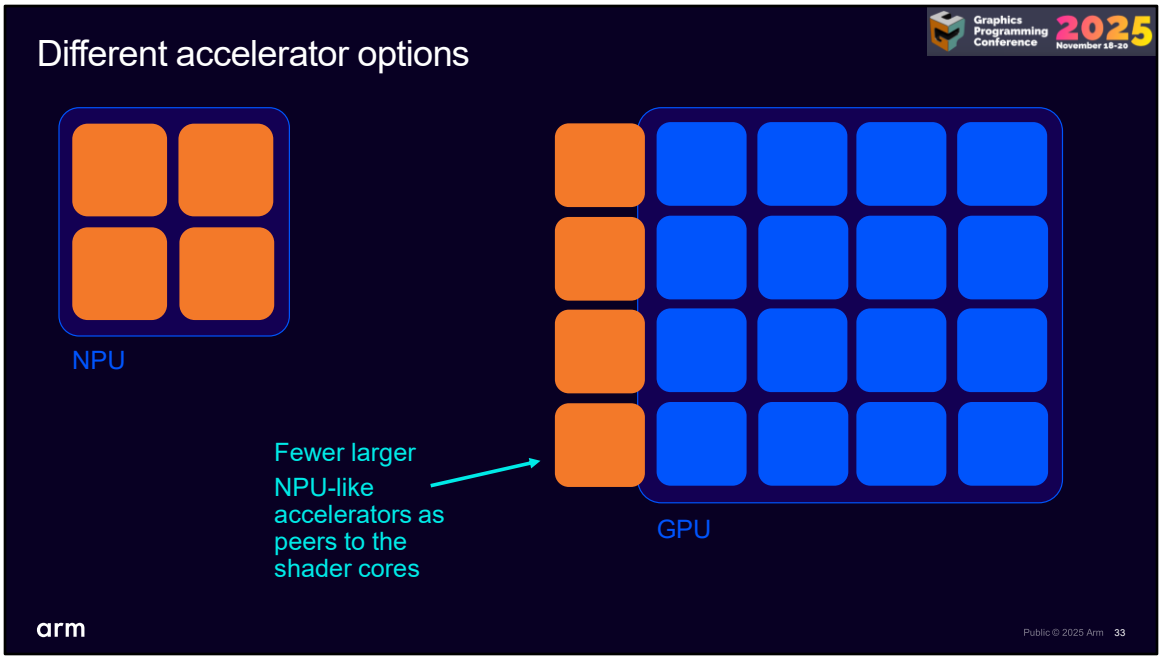
This is what many vendors, particularly desktop vendors, are doing today. It's a fine-grained accelerator that re-uses the most amount of existing HW but isn't necessarily the most optimal accelerator architecture.



We could tweak it a bit though.

What if we add small NPU-like accelerators into some of the cores. This gives us better efficiency – it's now a more coarse-grained "NPU-like" in capabilities and operation – and we can vary the amount needed independently from the number of GPU cores. In practice is a dispatch-sized rather than thread- or warp-sized accelerator which affects the programming model, but brings greater throughput and efficiency.

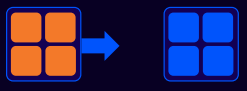
For full disclosure, this is what Arm is doing – it's the neural technology we first announced at SIGGRAPH 2025.



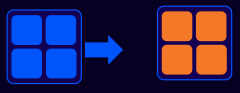
It’s not the only option though. We could instead make NPU-like cores as peers to the GPU cores.

A range of accelerators

- Multiple options, different trade-offs
- Matmul is not the solution for everyone, particularly not in mobile
- Motivations
  - AI improves with scale – need the ability to reach high perf/W points
  - At least on mobile, maximising performance is a battle with 1) memory bandwidth, 2) power 3) area
- NPU-like architectures are strong on these point
- Some converge between these architectures seem inevitable
- We will need API(s) that span a range of options



NPU tech is coming to GPUs…



…and GPU tech is evolving to be more like NPUs

arm

The important takeaway is there is not one way to do things.

ML is “everywhere” and the “matmul” style extensions to GPUs we not the only options we will see in practice.

The motivation is largely peak throughput and power efficiency

Power and bandwidth are the main limiting factors.

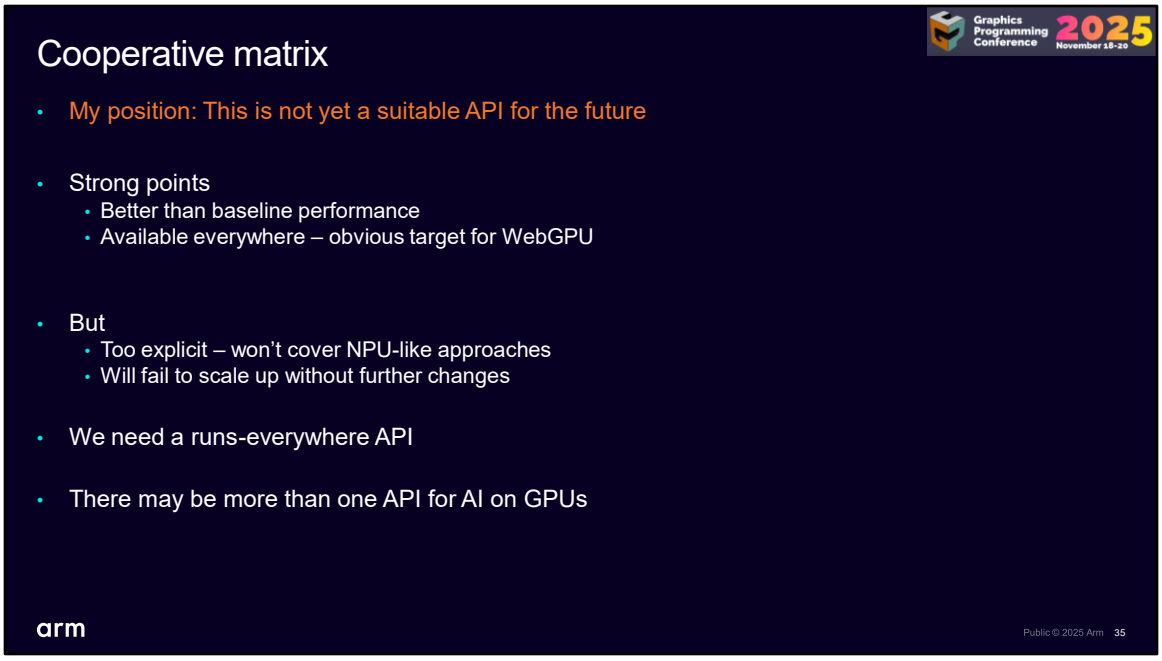
NPU architectures are a lot more efficient in both respects

Particularly in terms of increasing the potential macs/byte

NPU technology is coming to mobile

But also, we can observe that in the search for greater efficiency, GPUs are also evolving to be more like NPUs

We will need API options that span the range – possibly more than one option



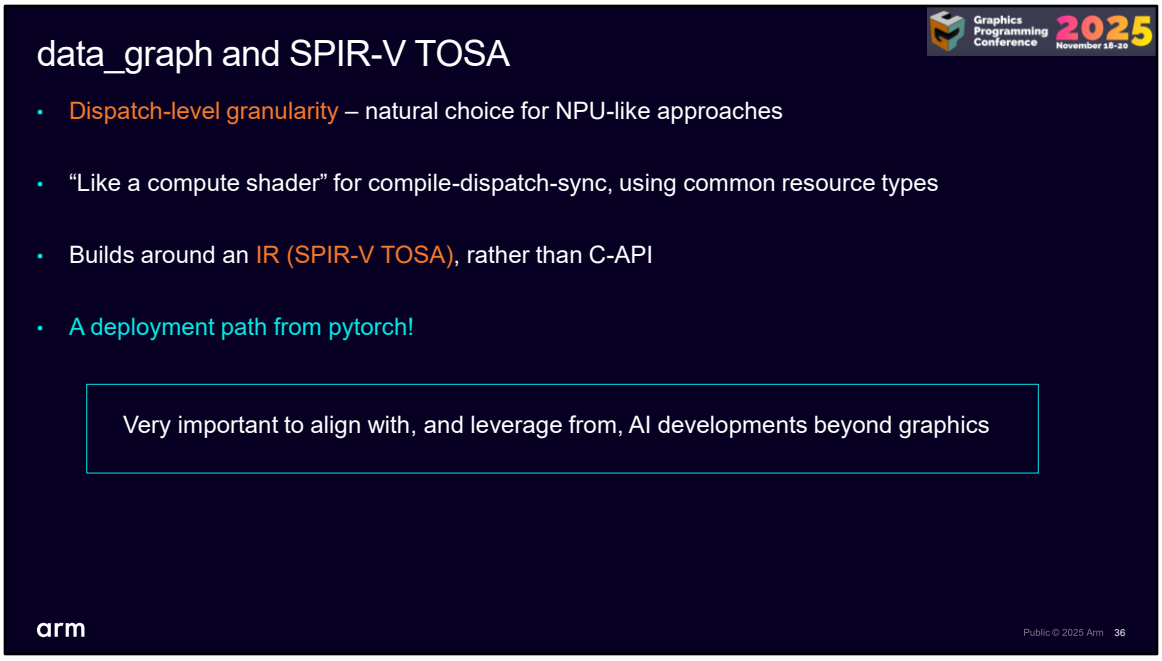
Looking at a couple of API options, an obvious one to look at is coop matrix.

Its strength lies in the fact it can be supported everywhere and will deliver better-than-baseline performance fairly reliably.

However, it is also unnecessarily explicit about how calculations are done in a way that would like it from scaling up or covering other HW.

Getting better macs/byte depends upon sharing data. If you are too explicit about how much sharing is allowed, or what other data movement tricks you might want to pull, then the API will become the barrier to better perf.

We need some evolution of coop matrix that is more forward looking but has the same runs-everywhere property.



Another option is the proposal Arm has been pushing.

This is a dispatch-level API but designed to work exactly like a compute dispatch. It provides the necessary freedom to the implementation to scale up the computation and focuses programmability on an IR rather than a C or shader API. The move to IR-based APIs is an important one.

One thing that unlocks is a deployment path from pytorch. This is critical: There is a lot of effort going on around AI and infrastructure for designing/training/deploying networks. The graphics community should be tapping into this, and a key way of doing so is to build on top of the MLIR ecosystem. Extending SPIR-V is a natural choice for doing that in Vulkan.

A fast deployment path is key to iteration, and fast iteration is key to success in basically all aspects of engineering.

What else could we have discussed?

- What other neural graphics use cases are viable?
- Neural shading / coop_vector
- Gaussian splats / alternative scene representations
- AI's impact on the way we create / implement / debug graphics

- I look forward to hearing your thoughts throughout the rest of the conference!

arm

Thanks! Q&A?

Thanks for the reviews / suggestions

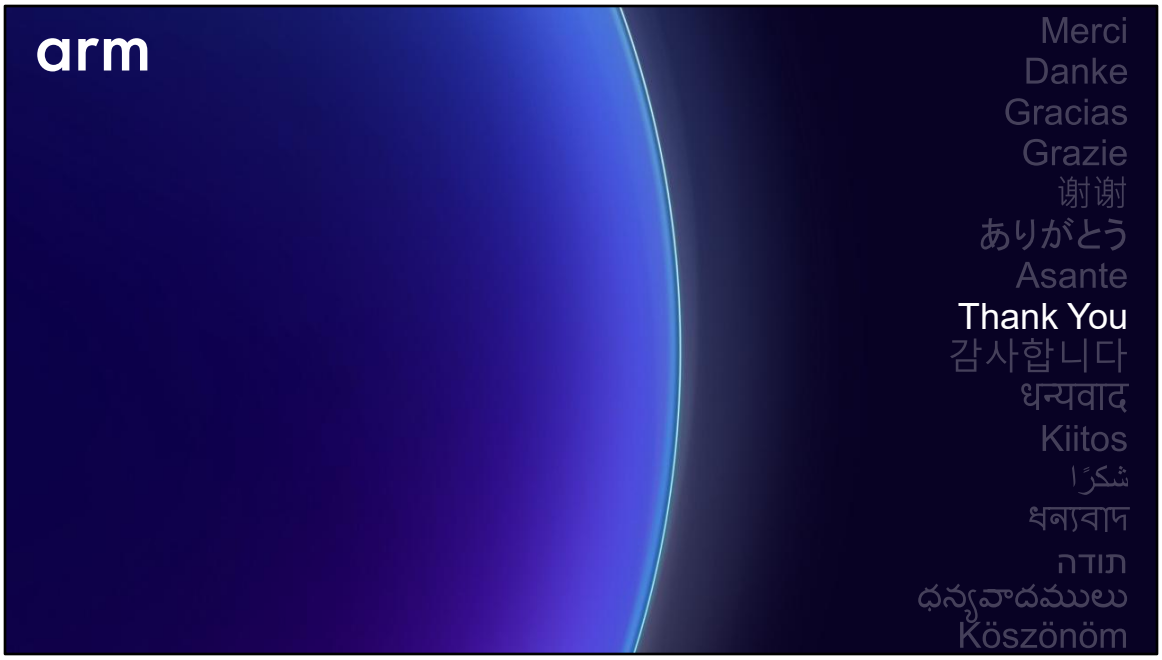
Their views may differ from those expressed in this presentation

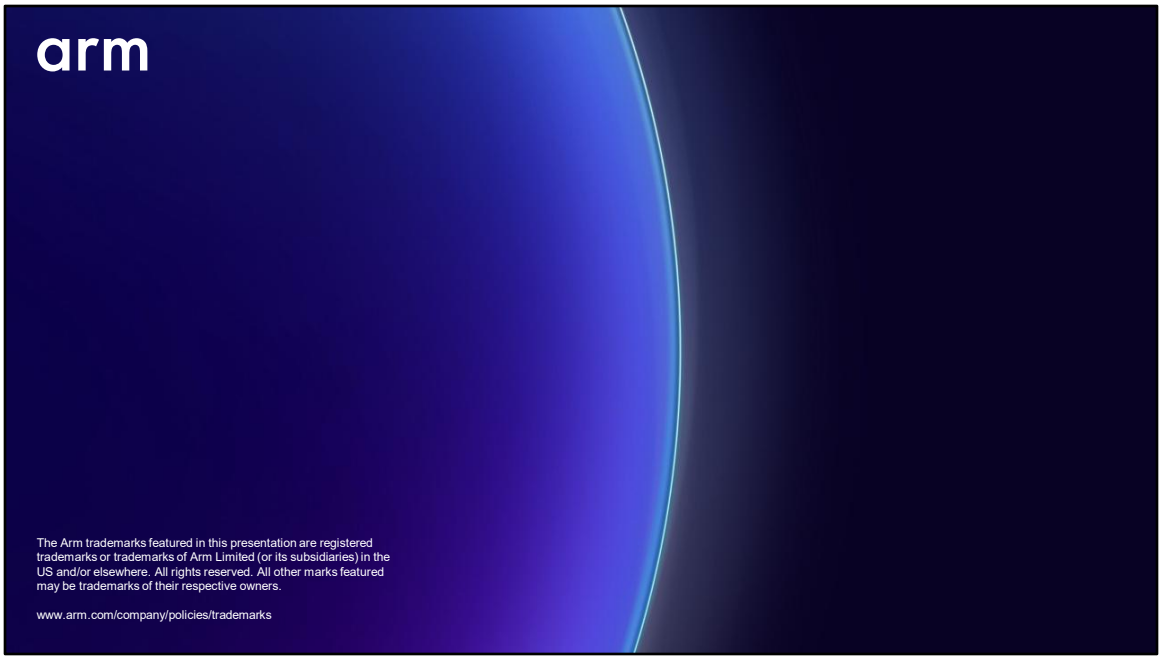
- Jasper Bekker
- Matthaus Chajdas
- Jan-Harald Fredriksen
- Liam O'Neil

Things we touched on

- Lessons learned from the evolution, and landing of, technology into graphics APIs
- A future beyond triangles
- The incoming impacts of AI on graphics, the importance of NPU-like designs and API challenges

arm


arm
Merci
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Thank You
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు
Köszönöm


arm
The Arm trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.
www.arm.com/company/policies/trademarks