HOT3D: RAY TRACING WITH IMAGINATION

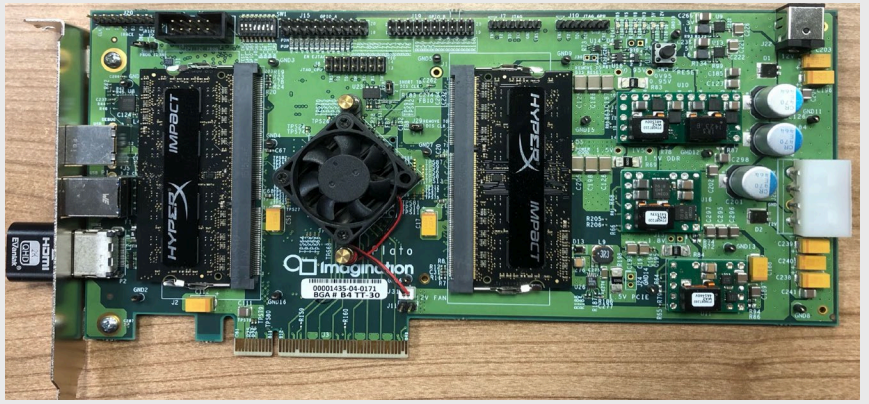
FROM LUX & PARSEC, THROUGH WIZARD TO PHOTON

June 2023

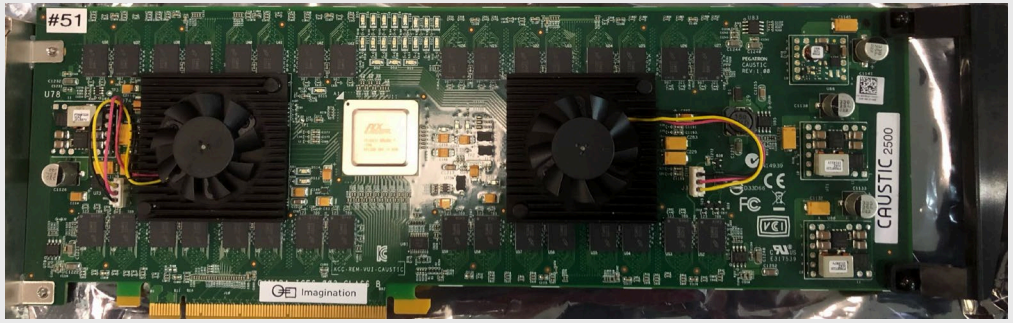
Simon Fenney

OVERVIEW

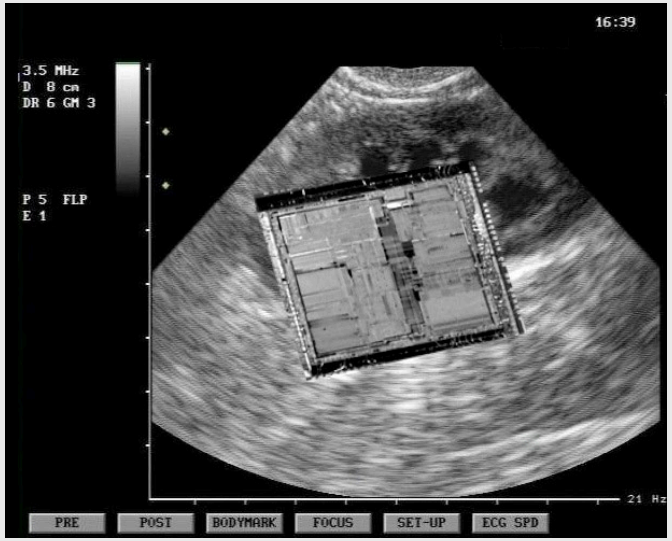
- Some history : “Lux”, “Parsec” and “Wizard/Plato”
- A little on the upcoming Photon
- Common DNA
- What has changed.



Wizard/Plato



Parsec / Caustic 2500



Photon

What does Imagination do?

Graphics, AI, CPU and more.

Key Products:

- ✓ Graphics and GPU compute scaling across all markets
- ✓ High-performance CPU
- ✓ Dedicated AI hardware

Business Model:

- ✓ Licensing of IP to customers and royalties
- ✓ Aligns us with customer success

Global Presence:

- ✓ UK-based global supplier with strong sales across the US, East Asia and Europe with Sales and R&D teams across several markets

KEY FOCUS AREAS

Mobile & Consumer

Automotive

Data center & Desktop

CPU

Some history of Imagination/PowerVR & Caustic Graphics

Going back in time…

At HPG 22 Aaron Lefohn presented "*If I had a DeLorean*"

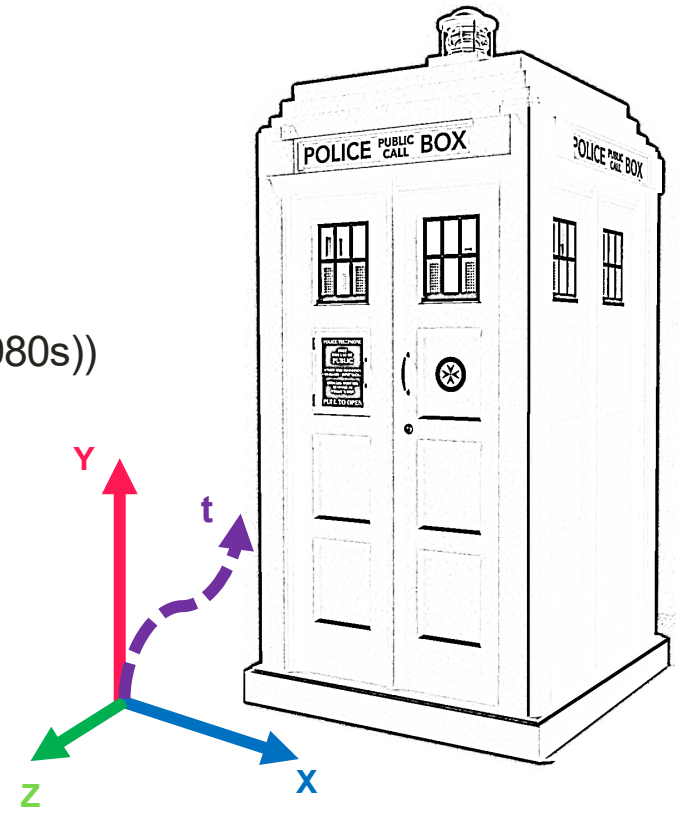
…But for ray tracing, perhaps an **AA*P*B** is more appropriate….

PowerVR started in 1992

- Project within Imagination (previously Videologic, nee Teletype (formed ~1980s))
- **Aim**: Efficient rasterised graphics hardware (TBDR)

Caustic Graphics started in 2006

- **Aim**: Efficient Ray Tracing
- Became part of Imagination in 2010



4 Generations – "Show and Tell"

From Professional 3D Visualisation through to Mobile

≈ 2009 "Lux"

- **For professional 3D visualisation**
- **25M *incoherent* rays/sec**
- **The FPGA predecessor of …**

≈ 2012 "PARSEC"

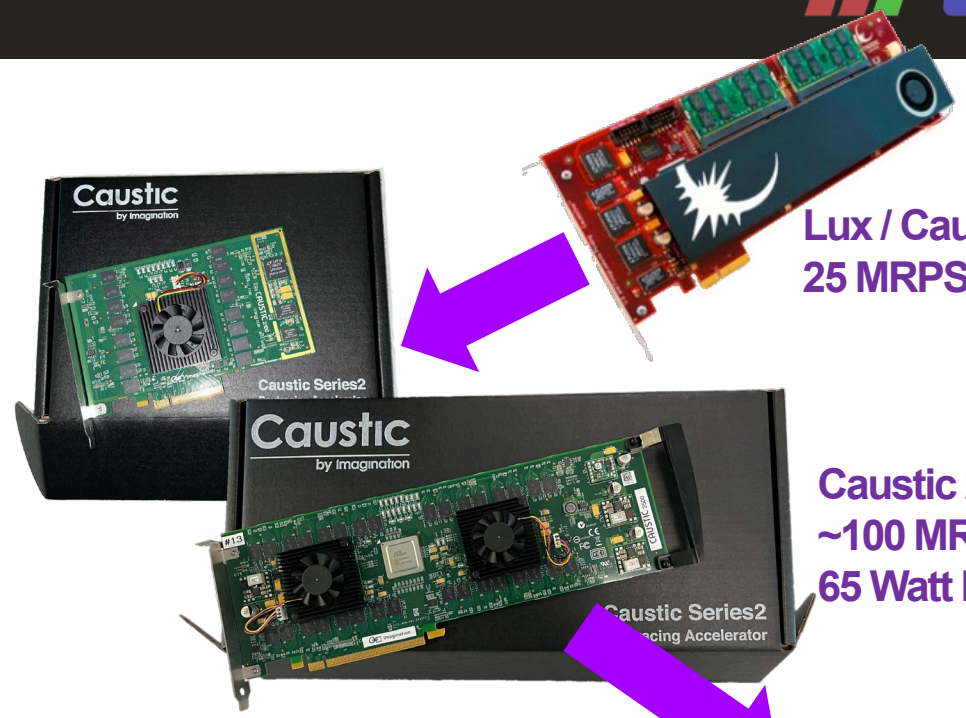
- **Parallel, BVH + Triangle accelerator**
- **90nm ASIC**
- **Shading on CPU**
- **Caustic 2100 & 2500**

≈ 2015 "Wizard" 1 and 2

- **Mobile (i.e. Battery) class device**

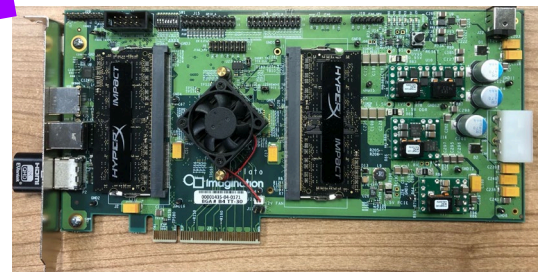
2021 "Photon"

- **Smaller and/or faster**



Lux / Caustic 1
25 MRPS

Caustic 2500:
~100 MRPS
65 Watt board



Wizard 1 / Plato
100+ MRPS
~2 Watt SOC

Photon:
~160 to 480 MRPS,
config dependent

Brief Diversion: “Incoherent rays”

Just as texture aliasing in rasterisation can “thrash your cache”

Primary rays – generally coherent

- **Camera rays from neighbouring pixels usually do similar things**
- **Go through many of the same nodes of the BHV**

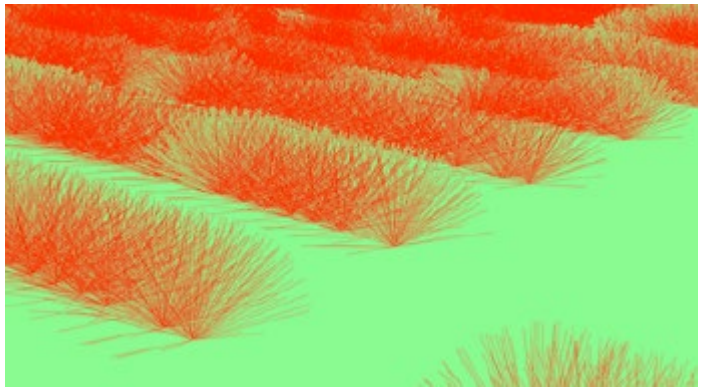
Secondary rays…

- **rapidly less and less coherent with each bounce**

Stochastic global illumination rays…

- **“Snowball’s chance” of being coherent.**

If handled naïvely, cache/memory systems may be suffering!



Common DNA

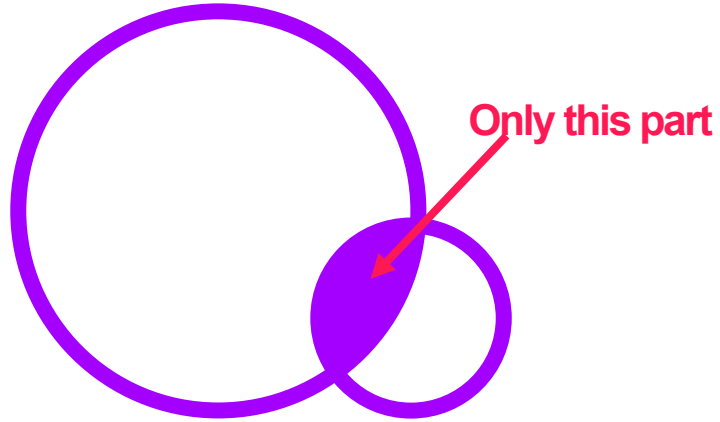
Broad aspects that have remained (mostly) in common

1. Automated HW BVH traversal & Triangle Testing

2. Ray/Geometry Coherency Gathering

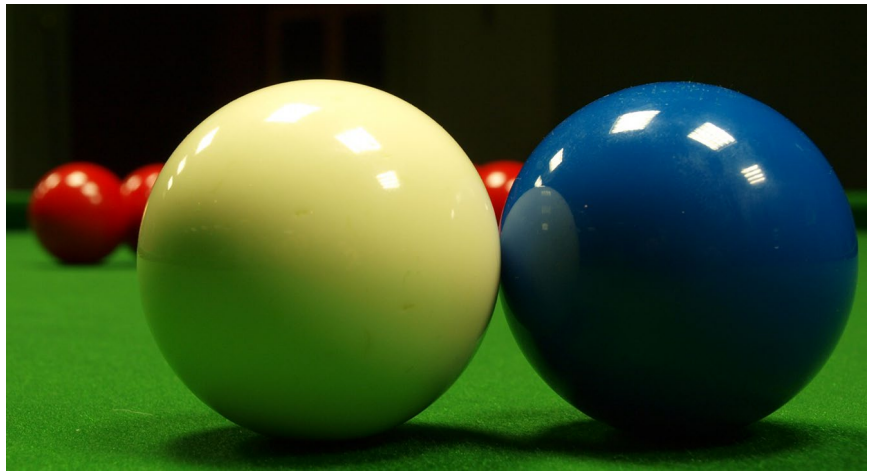
3. Shader Coherency

BVH Traversal



All generations traverse their AS/BVH automatically

- **Rays tested against Bounding Volumes and Triangles**
- **But can launch shaders for procedural objects**



But some differences

- **Lux & Parsec bounding volumes were intersections of 2 or 3 spheres**
  - Great for snooker scenes (could mark a sphere as “terminating”) …
  - …but trickier for BVH building. 😳
- **Wizard & Photon changed to AABBs.** 😀
  - Compressed format: each node defined 4 (or more) sub-boxes.
  - You don’t need full float precision.

Traversal Ray/Geometry Coherency Gathering

To avoiding thrashing memory system

Need to dealing with many arbitrary rays in the BVH

System has "N" packets of work

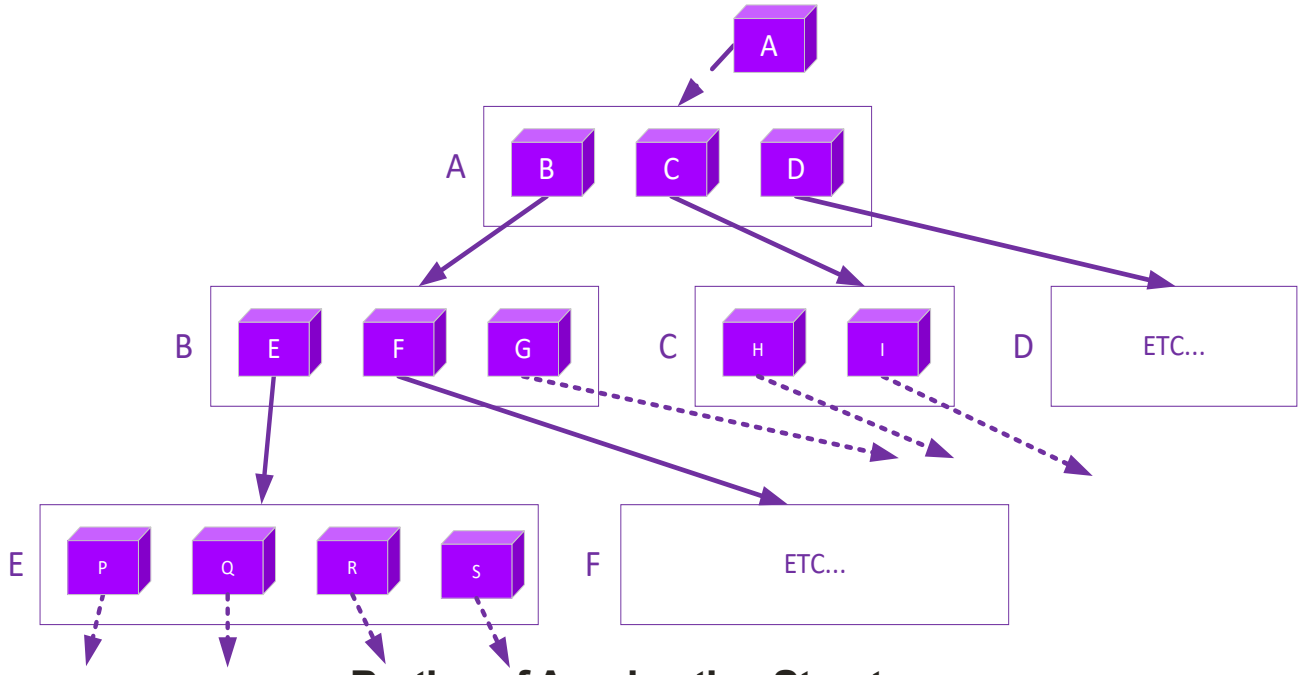
Each packet has:

- **Acceleration Structure Node ID {+instance}**
- **Rays to test vs node (eg child AABBs)**

Picks a packet. *e.g. if Node B*

- **Test rays *{5, 99, 787}* against children *{E,F,G}***
- **Add "hits" to corresponding packets**
  e.g. if 787 intersects E → add to E's packet.
- **Packets dynamically managed**

Triangle nodes can update ray extents/bounds (aka T-values)



Portion of Acceleration Structure

Node ID {+instance}	Rays
…	….
B	**5**, **99**, 787,
K	…
C	56, 2, **5**, 89, **99**, 128,
E	17,18, 19, +787

Shader Coherency

Do more sorting – be nice to your SIMD

Part of Imagination GPUs

Since PowerVR Series 2 (e.g. Dreamcast), GPU has reordered/sorted by "shading work"

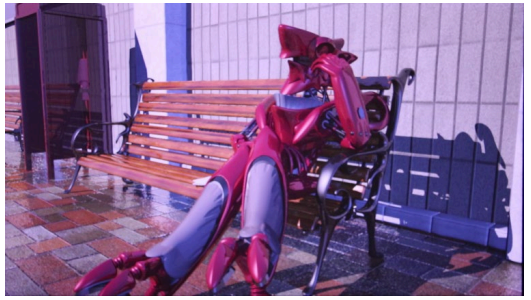
- **Initially just fragments for coherent texturing, but that then became shaders.**
- **It's automatic.**

Same with Ray Tracing.

- **Don't want low occupancy in shader units.**
- **PARSEC, Wizard & Photon *all* do sorting/gather by shader**

API Evolution : 1

To make HW useful, need an API



OpenRL: circa 2010

- **Lux, PARSEC, and Wizard 1**
- **Derived from OpenGL ES 2.0**
- **Full Ray Tracing model**
  - Frame Shaders (i.e. raygen/ camera)
  - Vertex Shaders (object placement)
  - Ray Shaders (for shading ray hits i.e. “closest hit”)

Hybrid rendering possible, but a little inelegant.

OpenGL ES extensions: circa 2015

- **Ray Tracing & Hybrid**

(Confession: Not *exactly* the HW I used)

A brief diversion: "Fire & Forget"

Ray Tracing is recursive, isn't it?

If you were writing a ray tracer in the late 80s…

- **To run on an array of Transputers…**
- **…in Occam … a parallel but *non-recursive* language**

Or, like Andrew Garrard♦ in the 90s, developing ray tracing hardware
(♦HPG 2018: "Cold Chips: ART's RenderDrive Architecture")

- **And you didn't want to stack shaders…**

Or Caustic, in the 00s, also wanting, *lots* of rays in flight…

THEN…

Fire & Forget: 2

It's a physically based model

You can use "Fire and Forget".

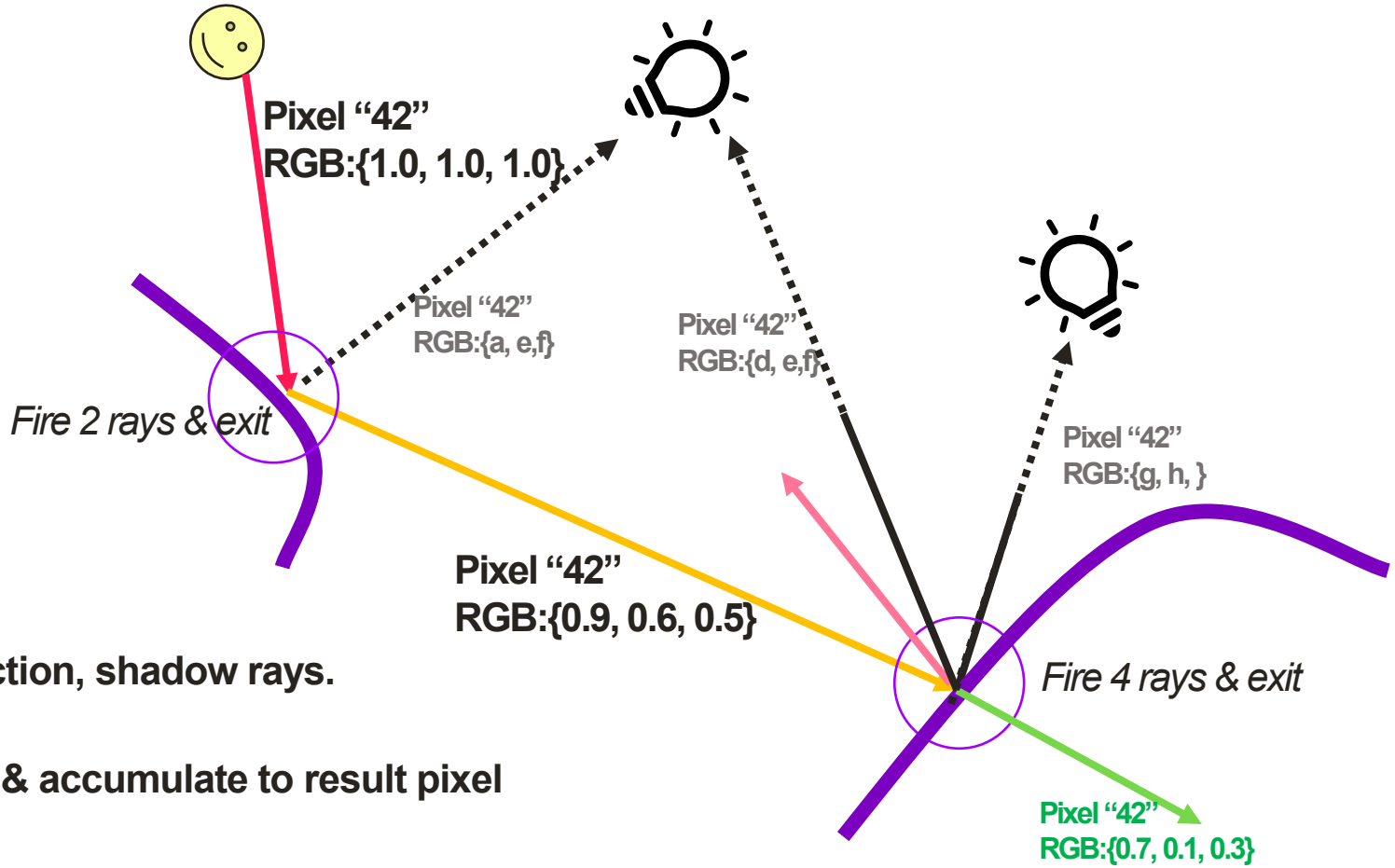
Each ray has

- **Address for result (eg pixel location)**
- **"RGB" attenuation weights**
- Optionally, recursion depth

When a ray hits a surface

- **Immediately 'fire' any new reflection, refraction, shadow rays.**
- attenuating ray weights appropriately.
- **Perhaps do any local/ambient illumination & accumulate to result pixel**
- **Exit the shader**

Other rays will do work at some later point



Back to API Evolution : 2

OpenRL/OpenGL extensions weren't getting traction

But then DXR and Vulkan Ray Tracing APIs arrive

- **Suddenly more interest**

Perhaps biggest change:

- **"CPU-like" recursive model - stacks your shaders**
- **"fire and forget" - fired or forgotten? 😢**

So photon is designed to support new APIs.

- **Both Ray Pipeline and Ray Query**
- **Instancing via TLAS (Top Level Accel Structure) + BLAS**

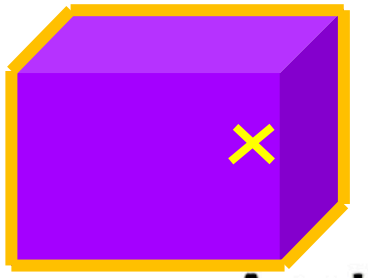
SO WHAT HAS EVOLVED?

Things learned along the way &

Making Photon smaller

Example changes: Ray AABB tests

Deep dive on an essential building block



Wizard and Photon use AABBs

Wizard used a Plücker AABB tester

(see McCombe, SIGGRAPH 2013 ,"*Ray Tracing is the Future and Ever Will Be*" https://dlnext.acm.org/doi/abs/10.1145/2504435.2504444)

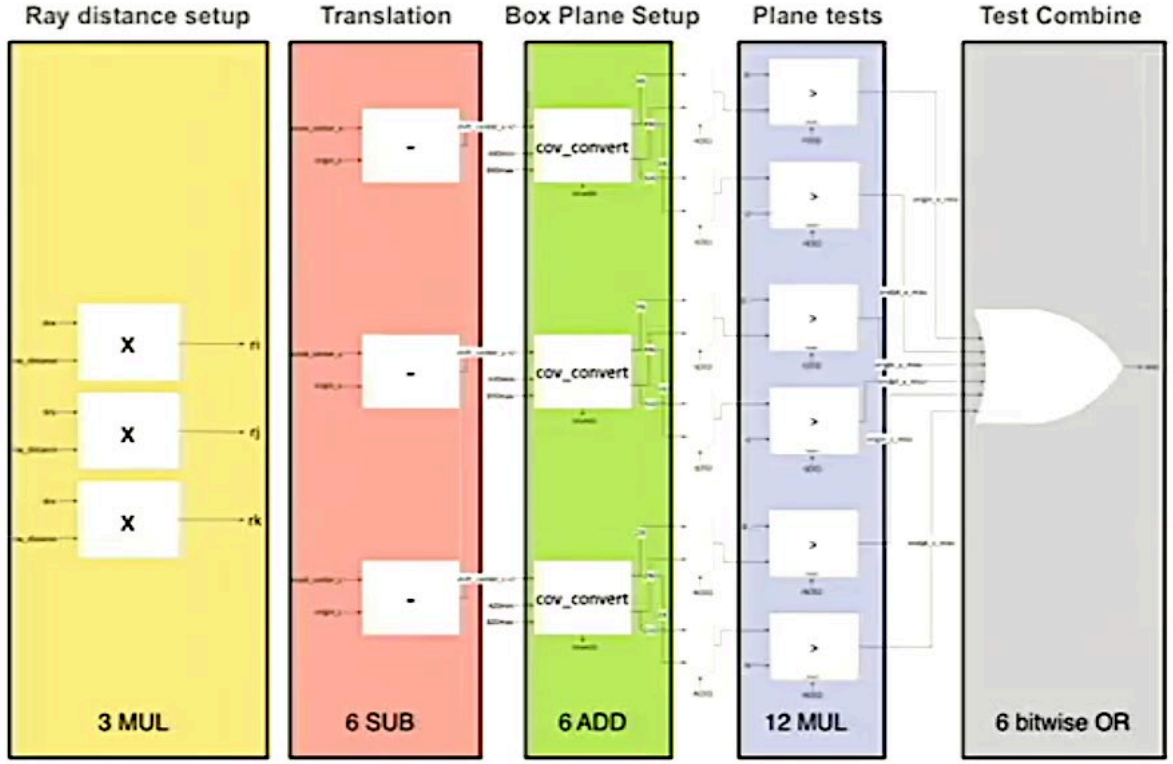
Tests ray against box's 6 silhouette edges – in parallel.

- Trades divisions for multiplies.
- Wizard: 15 Muls, 9 Adds or subtracts
- Adjusted to be 'conservative' (rounding up and down)
- + Added ray interval test

We wanted Photon to be smaller.

- **Could we do Ray-AABB with less HW?**

Analysis: Ray:AABB Test



Going back in time again…

"Back to the future"

Revisited other Ray-AABB tests

- e.g. Kay & Kajiya 86 or Smits 98/99.

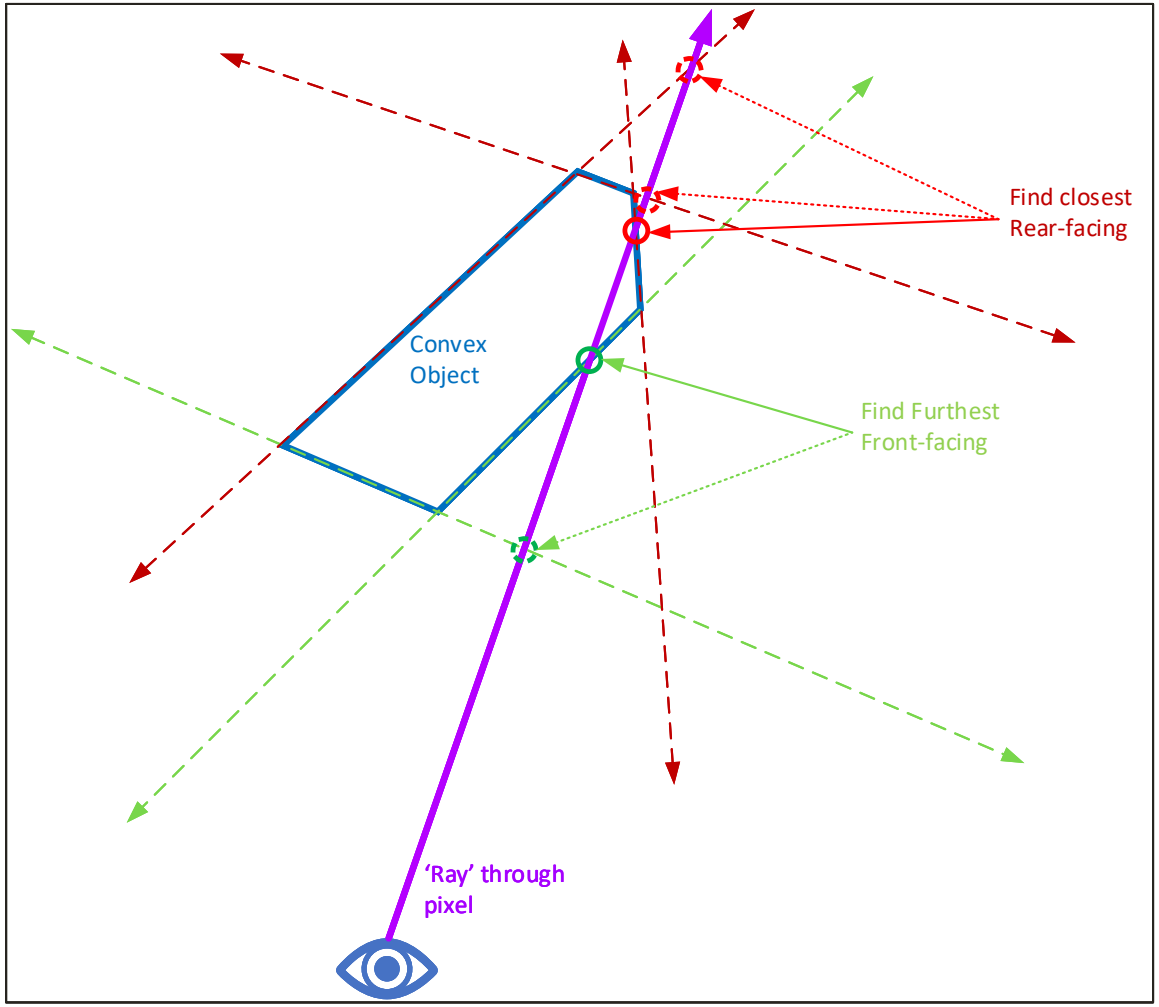
Reminded 'us' of PowerVR Series 1 (1992~1996)

Rendered 'convex objects'

- Which were built from intersection of planar half-spaces
- Graphics library gave hardware "front facing planes" then "rear facing".
- For each pixel, HW finds *furthest* front plane, then *nearest* rear plane.

Can we use this to do less work (i.e. less silicon)?

- though possibly with higher latency



Photon Ray–Box Plane tests

Testing AABBs with less

1. Like Wizard, translate AABB by ray origin.

2. Make ray direction "canonical".

- **Permute** both Ray & AABB axes *and* flip sign bits so that
- Ray direction is positive and has Z ≥ Y (≥ 0) & Z ≥ X (≥ 0)

3. Then "in effect" scale ray by 1/Z and compute inverses….

- Adjusted Ray "Direction" ← [Z/X , Z/Y, 1]
  - Needs 2 RCP *operations* and 2 muls.
  - Cost is amortised over multiple AABBs

"But what of division by zeros?".

We'll get back to that…

"And FP precision/consistency?"

We do the same for ray-tri tester, so they match behaviour.

Ray Box-Plane tests 2

Much of this was done in “lock down” – assume approximately correct

To be ‘safe’, AABB is “expanded” to include FP error of

- **the AABB testing &**
- **worst case error of Ray-Triangle tester**

Check T_{min}/T_{max} and Box Octants

- **Can do “early out”**

Compute ‘distances’ to Box XYZ planes

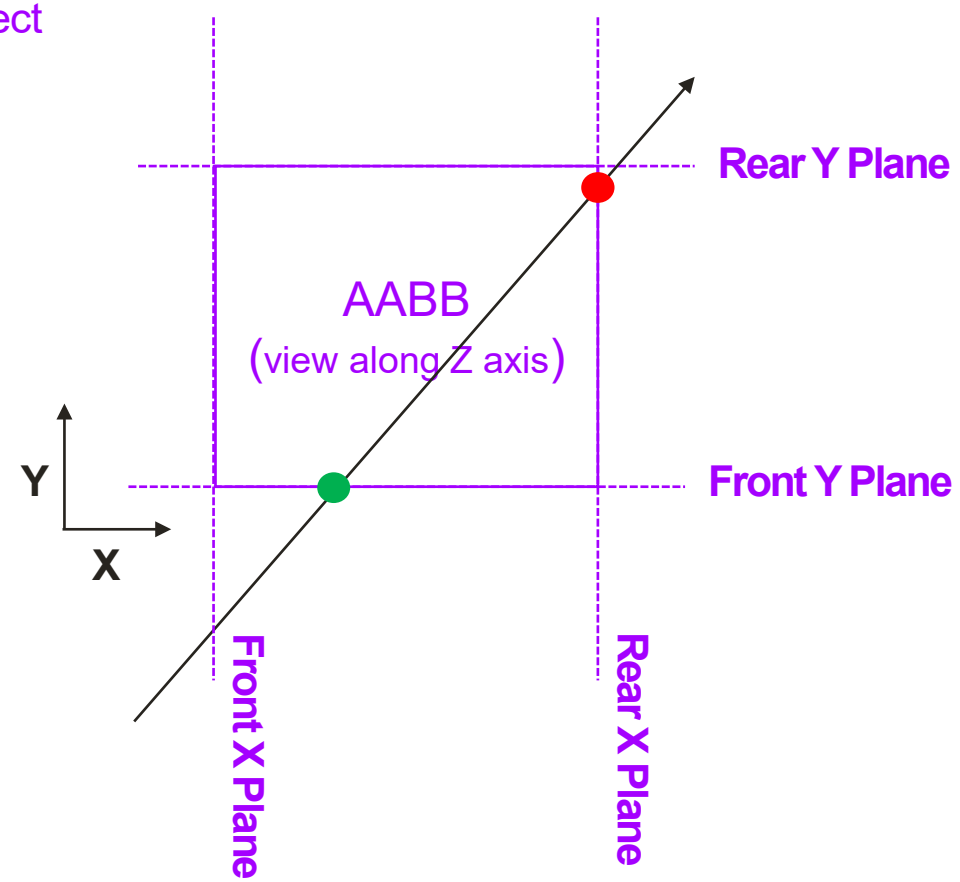
- **This needs 4 muls – Z is “free”**

Get Max of Front plane distances, Min of Rear

Reject if Front > Rear

- **bakes in a conservative safety margin**

Do conservative T_{min}/T_{max} tests



Dealing with those tedious “zeros”

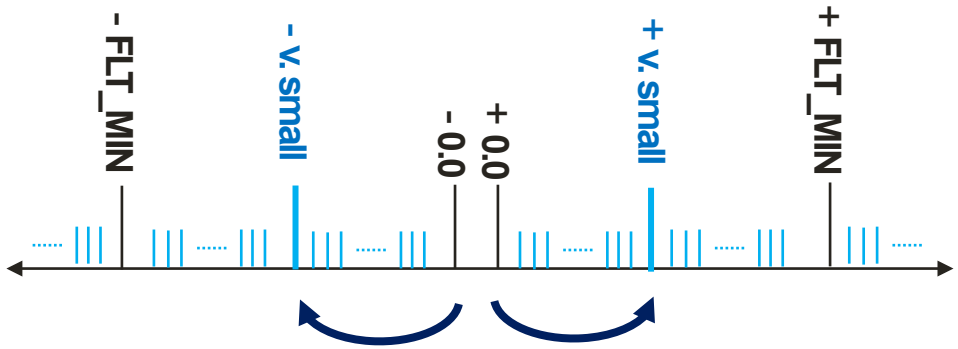
Joys of custom RTL

The Ray-AABB test is / must be conservative

We are specifying the functional unit so we can *start* with IEEE float…

And add more exponent bits.

- **Introduce special “infinitesimal” values (< FLT_MIN)**
- **Zeros remapped.**
- **Avoids the messy, special cases!**



For AABB testing, 23 bits of *mantissa* is overkill.

- **Reduce precision to save area.**

Some other Photon changes relative to Wizard 1

Added: Instance Transform Unit

- **Matrices used to position instances**
  - AKA BLASs (Bottom Level Acceleration Structures)
- **Transform ray by inverse instance transformation**
- **But we do in two stages.**
- **Lose FMA convenience, but gain accuracy.**

$$M_{instance} = \begin{bmatrix} R_{00} & R_{01} & R_{02} & T_x \\ R_{10} & R_{11} & R_{12} & T_y \\ R_{20} & R_{21} & R_{22} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{inst\_inv} = (M_{instance})^{-1} = \begin{bmatrix} S_{00} & S_{01} & S_{02} & U_x \\ S_{10} & S_{11} & S_{12} & U_y \\ S_{20} & S_{21} & S_{22} & U_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{inst\_inv} = \begin{bmatrix} S_{00} & S_{01} & S_{02} & 0 \\ S_{10} & S_{11} & S_{12} & 0 \\ S_{20} & S_{21} & S_{22} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -T_x \\ 0 & 1 & 0 & -T_y \\ 0 & 0 & 1 & -T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Removed: Streaming Hierarchy Generator

- **Wizard had a hardware BVH builder (SHG)**
- **This built AS 'on the fly' as the vertex shader streamed out batches of triangles.**
- **But any silicon in SOCs is 'money' so Photon instead uses 'compute'**

Ray Triangle Testing

"Honey, I shrunk the DTTU"

Wizard & Photon both test rays vs triangle *pairs*

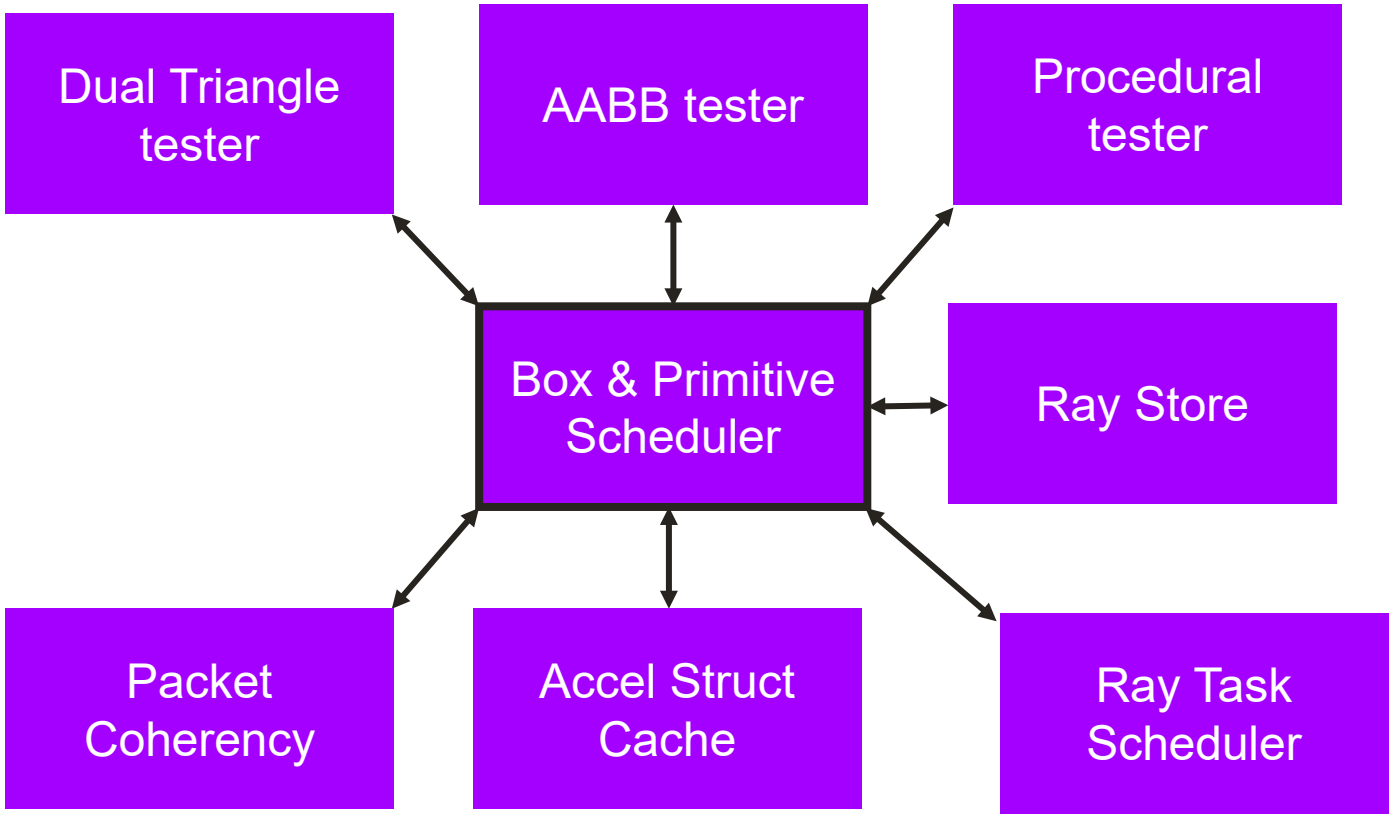
- **"Dual Triangle Tester Unit"**

Using 'tricks' learned from Photon's AABB tester…

- **Reduced the number of maths ops (relative to Wizard).**
- **That's possibly a whole talk in itself..**

Hardware Unit : “RAC”

Ray Acceleration Cluster



BUT THIS IS GRAPHICS

So we need some pictures…

Wizard in action

Running on single Wizard 1 board

Linux Wizard Demo machine

Mobile class GPU

Feel free to hum your own accompanying music

“More Ray Tracing”

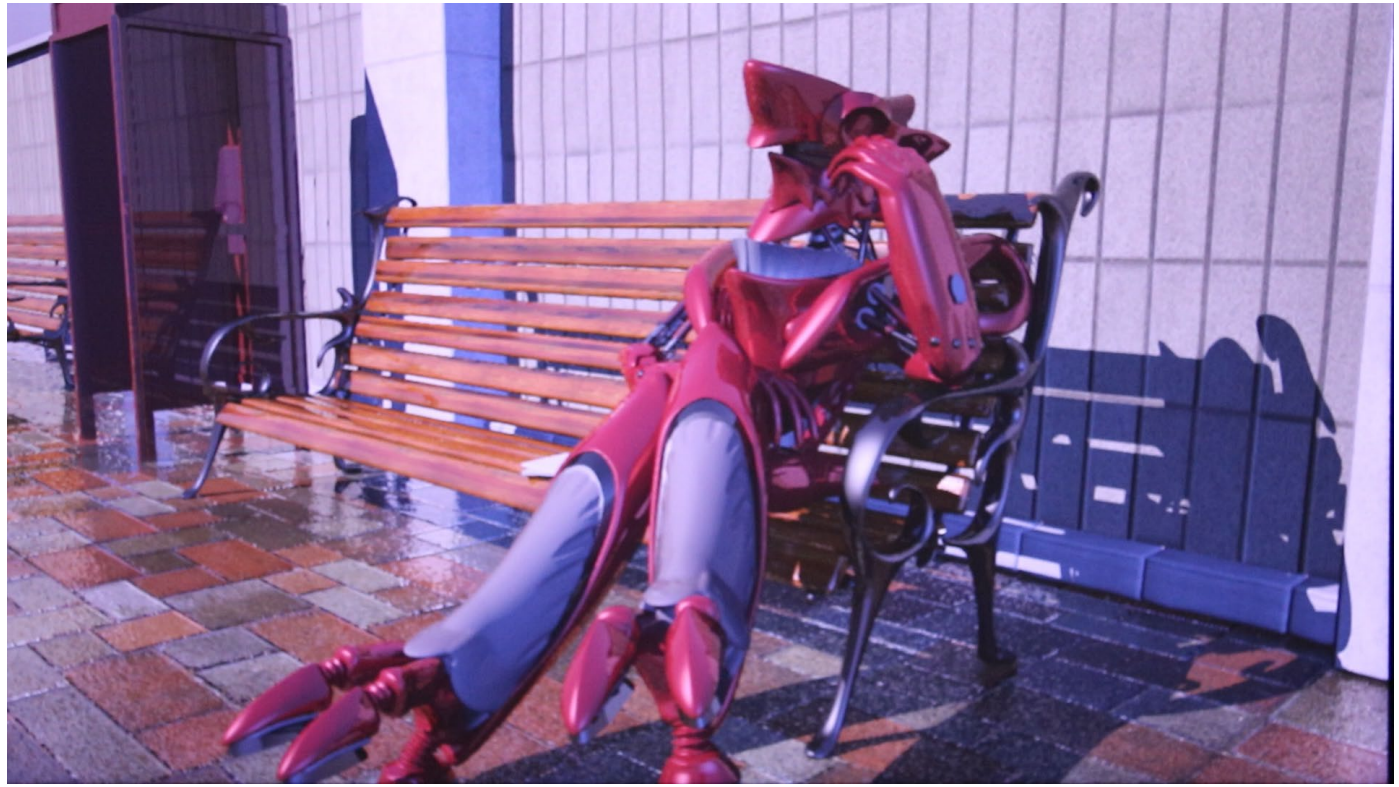
If the previous video wasn’t obvious enough – write it in big glass refractive letters

“Robot”

Wizard: Comparison with other contemporary HW

See “Tom’s Hardware”:

https://www.youtube.com/watch?v=Fz6AUj2PY9c



Photon Related Demos

No physical hardware so these are sims

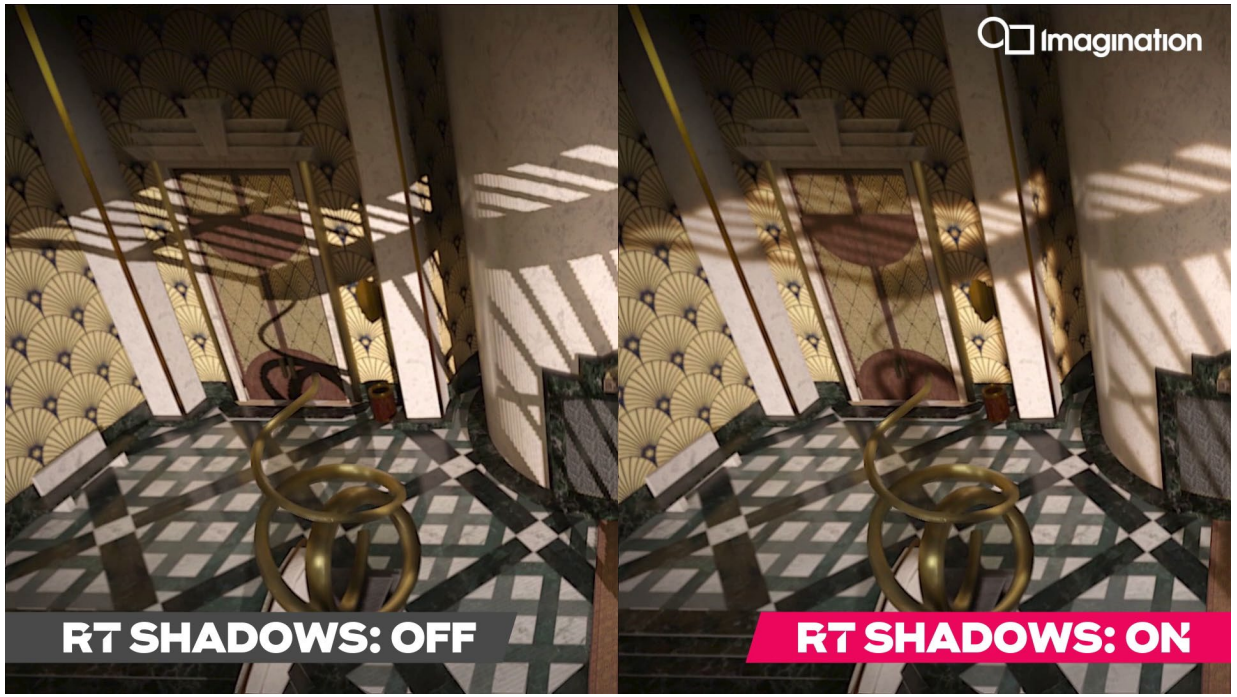
Many initial “ray tracing apps” on mobile will probably be hybrid renderers.

…so this is a hybrid, RT light-probe GI demo, I believe aimed at lower end configurations.

Photon demos (sim)

Another Hybrid example – launch rays from G-buffer

***Shadows*: shadow maps vs "just fire rays"**
***Reflections*: Subtle, but floor, 'painting' and 'diver' all reflect the scene**



Useful links

Photon & Imagination University Programme training materials

IMAGINATION

RAY TRACING

High performance, desktop-quality ray traced visuals on mobile

IMAGINATION

UNIVERSITY PROGRAMME

Supporting teachers around the world

Acknowledgements

Colleagues past & present

Special thanks to

- **Andrew Garrard**
- **Dan Cooney**
- **James McCombe**
- **Luke Peterson**
- **Panos Velentzas**
- **Robin Britton**
- **GPU Research Team**
- **(And to others I've forgotten to mention)**

…mostly for filling a few gaps in my knowledge/memory.

Imagination

THANK YOU FOR LISTENING

QUESTIONS?