SIGGRAPH2015
Xroads of Discovery

The 42nd International Conference and Exhibition on Computer Graphics and Interactive Techniques

The Rendering Pipeline - Challenges & Next Steps

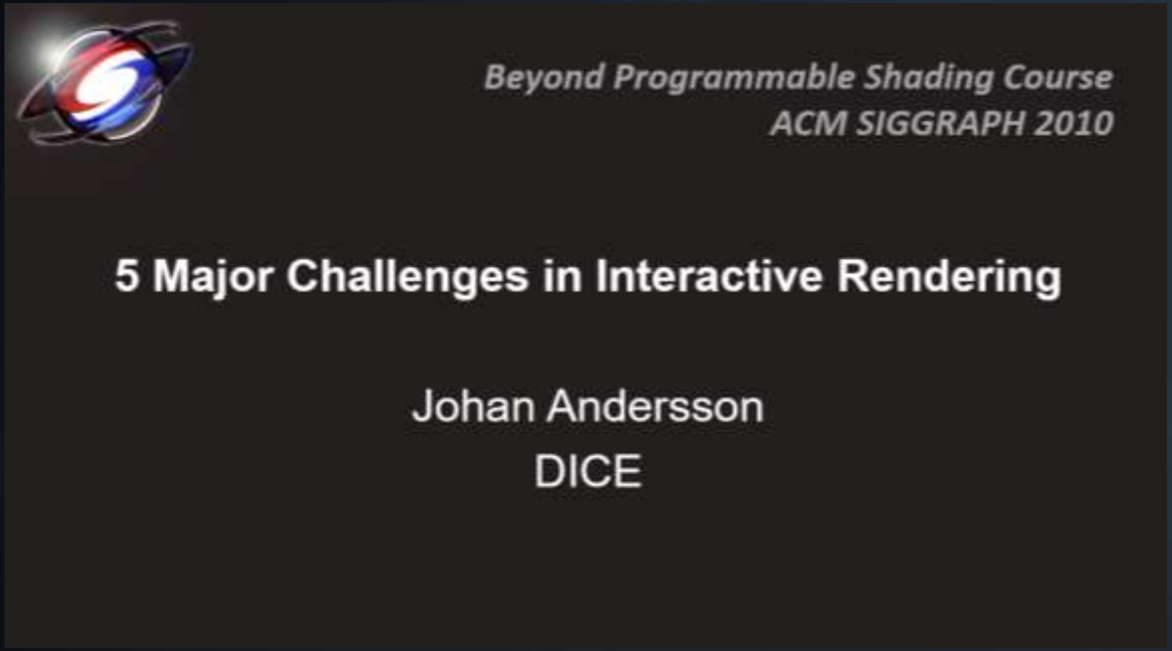
JOHAN ANDERSSON

ELECTRONIC ARTS

Intro

- What does an advanced game engine real-time rendering pipeline look like?
- What are some of the key challenges & open problems?
- What are some of the next steps to improve on?
- From both software & hardware perspectives

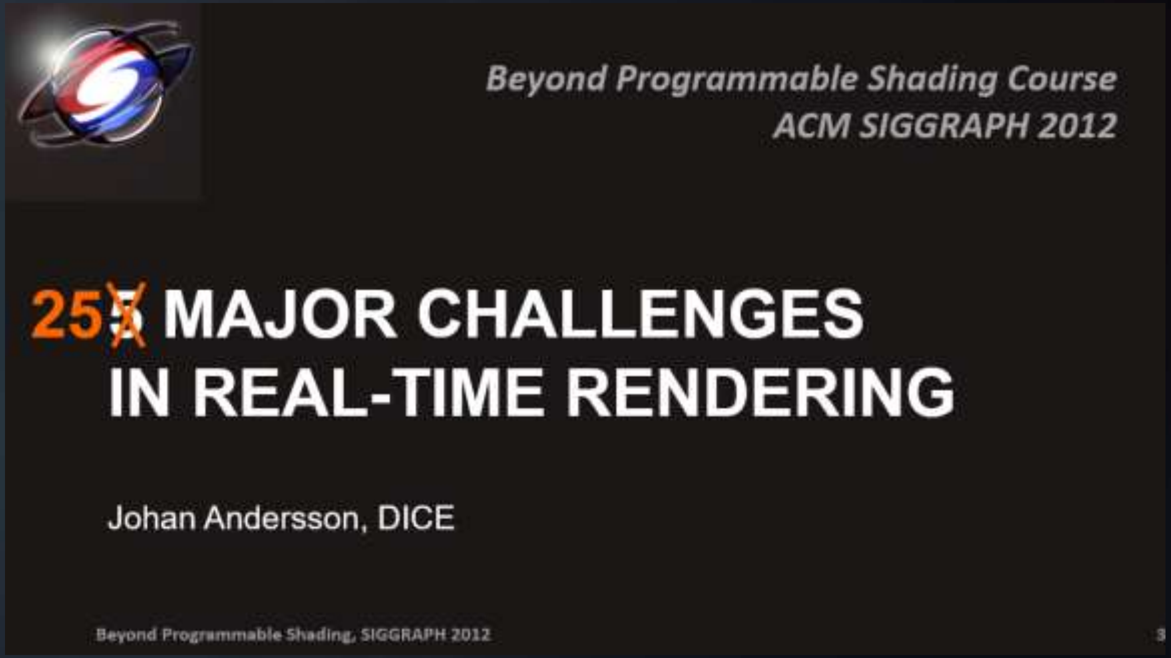
Previous talks



Beyond Programmable Shading Course
ACM SIGGRAPH 2010

5 Major Challenges in Interactive Rendering

Johan Andersson
DICE



Beyond Programmable Shading Course
ACM SIGGRAPH 2012

25~~5~~ MAJOR CHALLENGES IN REAL-TIME RENDERING

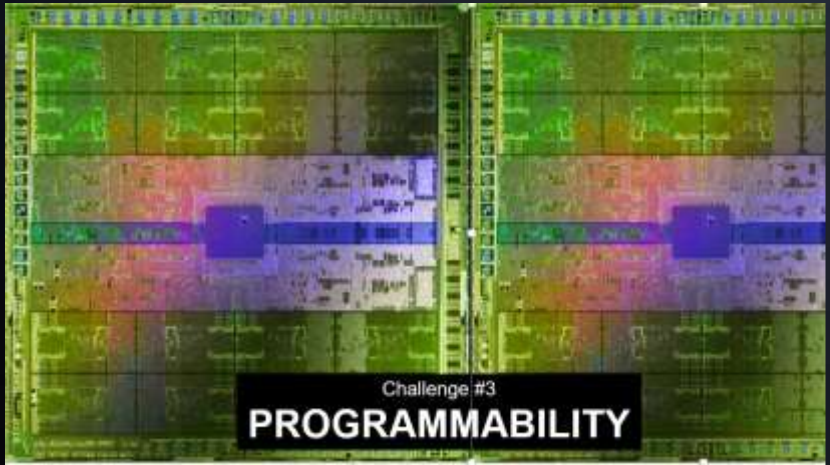
Johan Andersson, DICE

Beyond Programmable Shading, SIGGRAPH 2012 3

2010 & 2012 challenges

Challenge #1
CINEMATIC IMAGE QUALITY

Challenge #2
ILLUMINATION


Challenge #3
PROGRAMMABILITY

Challenge #4
PRODUCTION COSTS

Challenge #5
SCALING

Long term goal:

Photo-realistic rendering at 1W

Improvements since 2010 & 2012

- Image quality & authoring: massive transition to PBR
- Reflections: SSR and perspective-correct IBLs
- Antialiasing: TAA instead of MSAA
- Gen4 consoles (PS4 & XB1) as new minspec
- Compute shader use prevalent – create your own pipelines!

Improvements since 2010 & 2012 (cont.)

- New explicit control APIs
  - Mantle, Metal, DX12, Vulkan
  - Well needed change & major step forward
  - Not much improvements on compute & shaders

- Programmability
  - Conservative raster, min/max texture filter
  - "Need a virtual data-parallel ISA" -> SPIR-V!
  - "Render target read/modify/write" -> Raster Ordered Views
  - Sparse resources

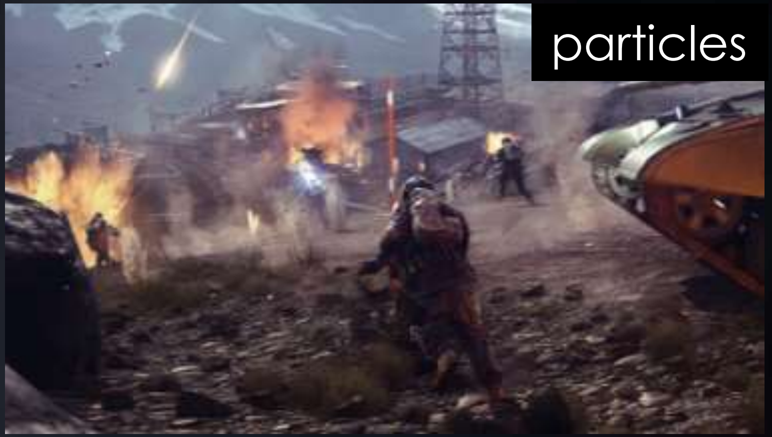
Pipeline of today – key themes

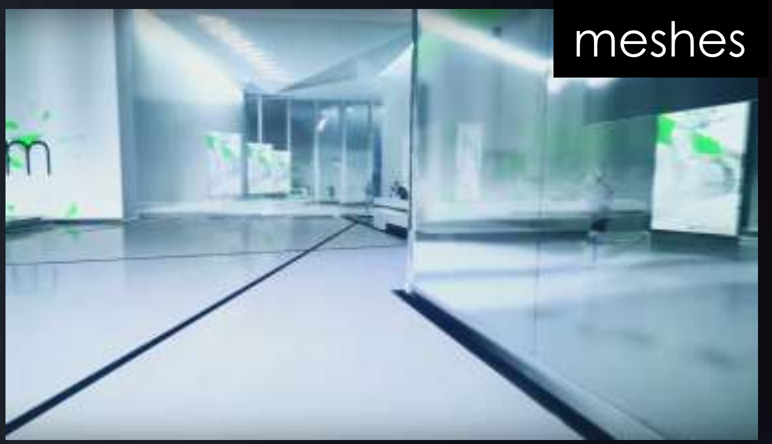
- Non-orthogonality gets in the way – can we get to a more unified pipeline?
- Complexity is continuing to increase
- Increasing quality in a scalable way

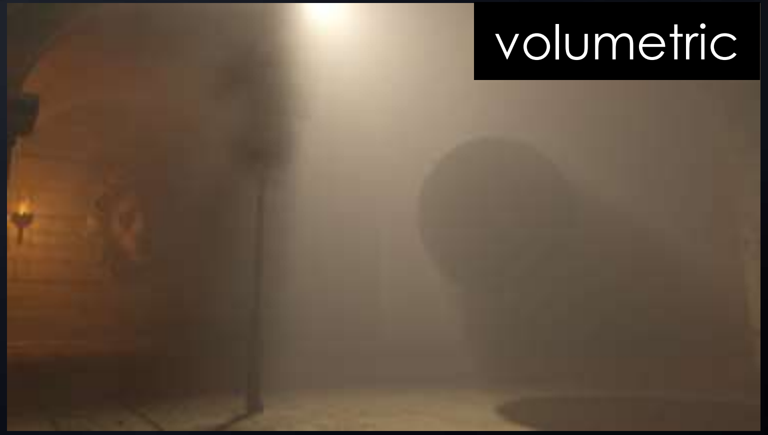
Getting to a more unified pipeline

Transparencies – sorting

- Can't mix different transparent surfaces & volumes
  - Particles, meshes, participating media, raymarching
  - Can't render strict front to back to get correct sorting
  - Most particles can be sorted, have to use uber shaders
- Contrains game environments
- Restricts games from using more volumetric rendering







Transparencies – sorting solution

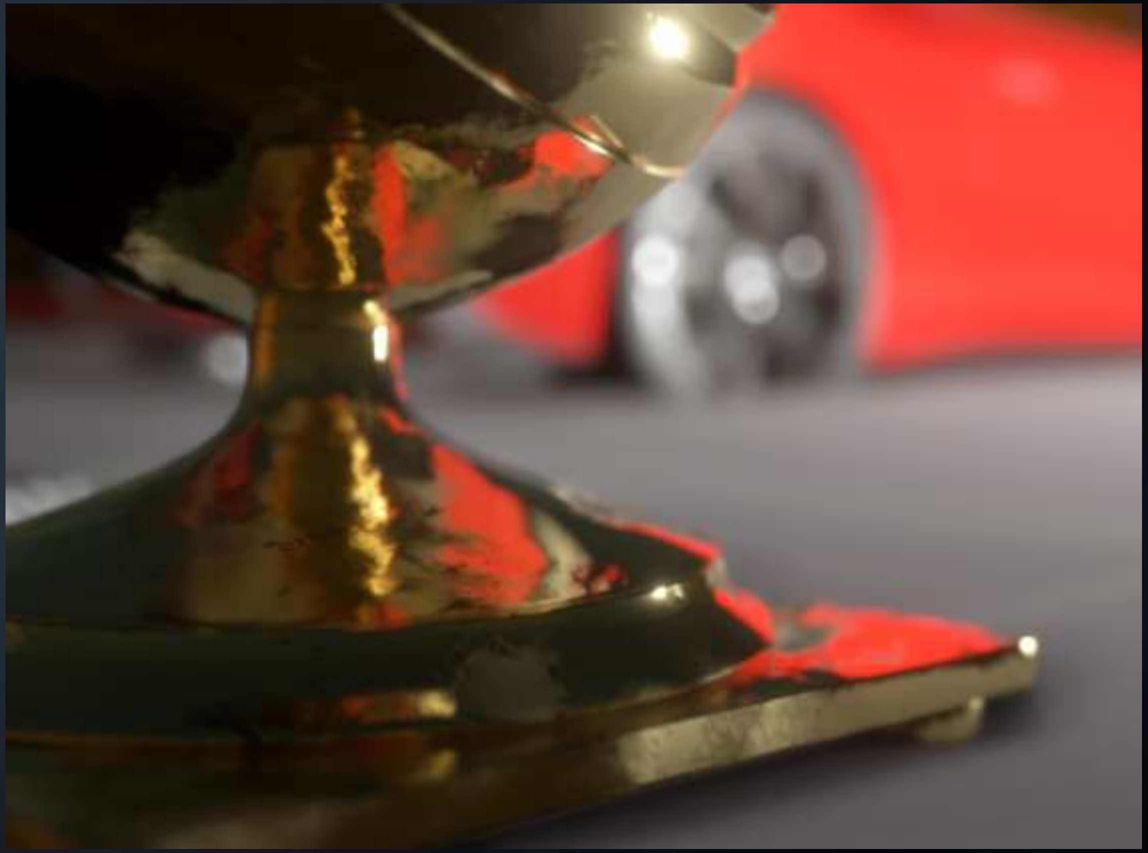
- Render everything with Order-Independent Transparency (OIT)
  - Use Raster Ordered Views (DX12 FL: Haswell & Maxwell)
  - Not available on consoles = most games stuck with no OIT

- Scalable to mix all types of transparencies with high quality?
  - Transparent meshes (windows, foliage): 1-50x overdraw
  - Particles: 10-200x overdraw
  - Volume rendering (ray-marched)

- Able to combine with variable resolution rendering?
  - Most particles & participating do not need to be shaded at full resolution

Defocus & motion blur - opaque

- Works okay in-game on opaque surfaces
  - Render out velocity vectors
  - Calc CoC from z
  - Apply post-process
- But not correct or ideal
  - Leakage
  - Disocclusion



Defocus & motion blur - transparencies

- Transparent surfaces are even more problematic
  - Esp. motion blur
  - Typically simulated with stretched geometry (works mostly for sparks)
  - Can only skip or smear everything with standard post-processes
  - Fast moving particles should also have internal motion blur

Defocus & motion blur - transparencies

- Blend velocity vectors & CoC for transparencies?
  - Feed into the post-process passes
  - Post-processes should also be depth-aware - use OIT approx. transmittance function
  - Still not correct, but could prevent the biggest artifacts
- Ideal: directly sample defocus & motion blur in rendering
  - But how? Stochastic raster? Raytracing?
  - Pre-filtered volumetric representations?

Forward vs deferred

- Most high-end games & engines use deferred shading for opaque geometry
    - Better quad utilization
    - Separation of material property laydown & lighting shaders

- Would like to render more as transparent, which has to use forward
    - Thin geometry: hair & fur
    - Proxy geometry (foliage) with alpha-blending for antialiasing

- But forward rendering is much more limiting in compositing
    - No SSAO
    - No screen-space reflections
    - No decal blending of individual channels (e.g. albedo)
    - No screen-space sub-surface scattering

Forward vs deferred (cont.)

- Can we extend either forward or deferred to be more orthogonal?

- Use world-space data structures instead of screen-space
  - Be able to query & calc AO, reflections, decals while forward shading
  - Texture shader to convolve SSS lighting
  - ***Massive*** forward uber shaders that can do everything

- Render opaque & transparent with deep deferred shading?
  - Store all layers of a pixel, including transparents, in a deep gbuffer
  - Unbounded memory
  - Be able to query neighbors (AO)
  - Be able to render into with blending (decals)

Rendering pipeline complexity

Rendering pipeline complexity

- Recent improvements that reduce complexity ☺
  - New APIs are more explicit – less of a black box
  - DX11 hardware & compute shaders now minspec
  - Hardware trend towards DX12 feature level

Rendering pipeline complexity

- Challenges:
  - Sheer amount of rendering systems & passes
  - Making architectural choices of what techniques & pipeline to use
  - Shader permutations & uber shaders
  - Compute shaders still very limiting – no nested dynamic parallelism & pipes
  - Mobile: TBDR vs immediate mode

Battlefield 4

Battlefield 4 rendering passes

- reflectionCapture
- planarReflections
- dynamicEnvmap
- mainZPass
- mainGBuffer
- mainGBufferSimple
- mainGBufferDecal
- decalVolumes
- mainGBufferFixup
- msaaZDown
- msaaClassify
- lensFlareOcclusionQueries
- lightPassBegin
- cascadedShadowmaps
- spotlightShadowmaps
- downsampleZ
- linearizeZ
- ssao
- hbaoHalfZ
- hbao
- ssr
- halfResZPass
- halfResTransp
- mainDistort
- lightPassEnd
- mainOpaque
- linearizeZ
- mainOpaqueEmissive
- mainTransDecal
- fgOpaqueEmissive
- subsurfaceScattering
- skyAndFog
- hairCoverage
- mainTransDepth
- linerarizeZ
- mainTransparent
- halfResUpsample
- motionBlurDerive
- motionBlurVelocity
- motionBlurFilter
- filmicEffectsEdge
- spriteDof
- fgTransparent
- lensScope
- filmicEffects
- bloom
- luminanceAvg
- finalPost
- overlay
- fxaa
- smaa
- resample
- screenEffect
- hmdDistortion

Architectural decisions

- Selecting which techniques to develop & invest in is a challenge
  - Critical to create visual look of a game
  - Non-orthogonal choices and tradeoffs
  - Difficult to predict the moving future of hardware, games and authoring
- Can be paralyzing with a big advanced engine rendering pipeline
  - Exponential scaling with amount of systems & techniques interacting
  - Difficult to redesign and move large passes
  - Can result in a lot of refactoring & cascading effects to the overall pipeline
  - Backwards compatibility with existing content
- Easier if passes & systems can be made more decoupled

What can we do to reduce complexity?

- A more unified pipeline would certainly help!
  - Such as with OIT
  - Or in the long term: native handling of defocus & motion blur

- Improve GPU performance – simplify rendering systems
  - Much of the complexity comes from optimizations for performance
  - Could sacrifice a bit of performance for increased orthogonality, but not much
  - We have real-time constrain = get the most out of our 16 ms/f (VR: 4 ms/f!)

- Raytrace & raymarch more
  - Easier to express complex rendering
  - Warning: moves to complexity to data structures and the GPU execution instead
  - Not practical overall replacement / unification
  - Use as complement – more & more common (SSR, volume rendering, shadows?)

What can we do to reduce complexity?

- Make it easier to drive the graphics & compute
  - CPU/GPU communication – C++ on both sides (and more languages)
  - Device enqueue & nested data parallelism
  - Increase flexibility, expressiveness & modularity of building pipelines
- Build a specialized renderer
  - Focus in on very specific rendering techniques & look
  - Typically tied to a single game
  - E.g. The Tomorrow Children, Dreams
- Build engines, tools & infrastructure to build general renderers
  - Handle wide set of environments, content and techniques
  - Modular layers to easily have all the passes & techniques interoperate
  - Shader authoring is also key

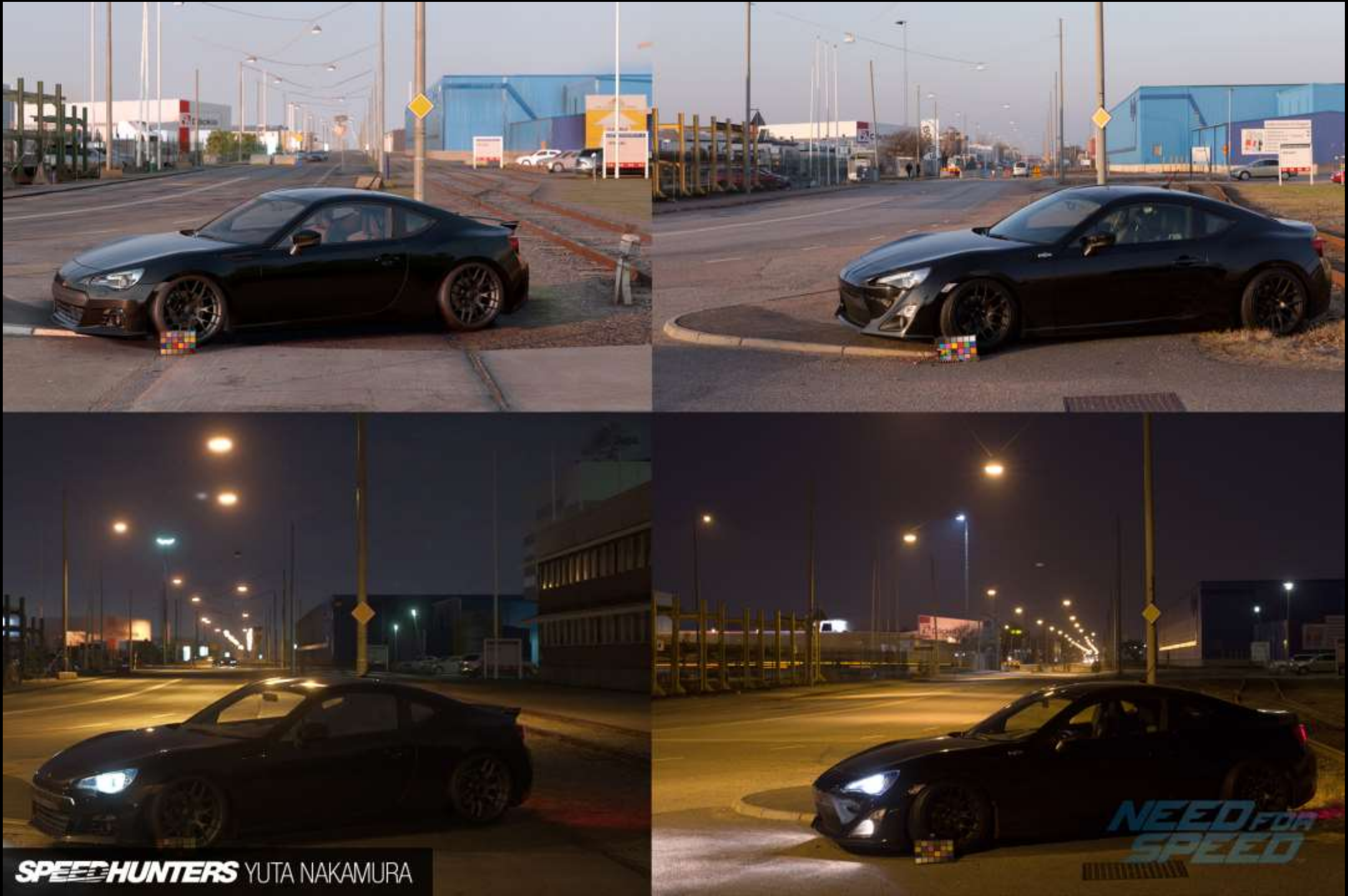
Uber shaders

- Example cases:
  - Forward shaders (lights, fog, skinning, etc)
  - Particles (to be able to sort without OIT) – want to use individual shaders instead
  - Terrain layers [Andersson07] – want to use massive uber shaders
- Why they can be a problem:
  - Authoring: Massive shaders with all possible paths in it, no separate shader linker
  - Performance: Large GPR pressure affects entire shader
  - Performance: Flow control overhead
- Classic approach: break out into separate shader permutations
  - Static CPU selection of shader/PSO to use – limited flexibility
  - Can end up creating huge amount of permutations = long compile/load times.
  - Worse with new APIs! PSO explosion

Uber shaders – potential improvements

- Shader function pointers
  - Define individual functions as own kernels
  - Select pointers to use per draw call
    - Part of ExecuteIndirect params
  - Ideal: Select pointers inside shader – not possible today
    - Optimization: VS selects pointers PS will use?
  - What would the consequences be for the GPU?
    - I$ stalls, register allocation, coherency, more?
- More efficient GPU execution of uber shaders?
  - Shaders with highly divergent flow & sections with very different GPR usage
  - Hardware & execution model that enables resorting & building coherency?

Scalable quality


SPEEDHUNTERS YUTA NAKAMURA
NEED FOR SPEED

Real-time rendering have gotten quite far!

- In order to get further, want to:

    1. Get that last 5-10% quality in our environments to reach photorealism

NFS photo reference

Real-time rendering have gotten quite far!

- In order to get further, want to:

1. Get that last 5-10% quality in our environments to reach photorealism

2. Be able to build & render new environments that we haven't been able to before

Glass houses!

Real-time rendering have gotten quite far!

- In order to get further, want to:

  1. Get that last 5-10% quality in our environments to reach photorealism

  2. Be able to build & render new environments that we haven't been able to before

Dreams (MediaMolecule)

Difficult areas

- Hair & fur
  - OIT, overdraw, LOD, quad overshading, deep shadows
- Foliage
  - OIT, overdraw, LOD, geometry throughput,
  - Lighting, translucency, AO
- Fluids
  - LOD & scalability, simulation, overall rendering
- VFX
  - Need volumetric representation & lighting
  - Related to [Hillaire15]

Difficult areas

- Hair & fur
    - OIT, overdraw, LOD, quad overshading, deep shadows
- Foliage
    - OIT, overdraw, LOD, geometry throughput,
    - Lighting, translucency, AO
- Fluids
    - LOD & scalability, simulation, overall rendering
- VFX
    - Need volumetric representation & lighting
    - Related to [Hillaire15]

Difficult areas

- Hair & fur
  - OIT, overdraw, LOD, quad overshading, deep shadows
- Foliage
  - OIT, overdraw, LOD, geometry throughput,
  - Lighting, translucency, AO
- Fluids
  - LOD & scalability, simulation, overall rendering
- VFX
  - Need volumetric representation & lighting
  - Related to [Hillaire15]

Difficult areas

- Hair & fur
  - OIT, overdraw, LOD, quad overshading, deep shadows
- Foliage
  - OIT, overdraw, LOD, geometry throughput,
  - Lighting, translucency, AO
- Fluids
  - LOD & scalability, simulation, overall rendering
- VFX
  - Need volumetric representation & lighting
  - Related to [Hillaire15]

Pompeii movie

Difficult areas (cont.)

- Correct shadows on everything
  - Including area lights & shadows!
  - Extra important with PBR to prevent leakage
  - Geometry throughput, CPU overhead, filtering, LOD

- Reflections
  - Hodgepodge of techniques today
  - Occlusion of specular critical
  - See Mirror's Edge talk [Johansson15]

- Antialiasing
  - See [Salvi15] next

Mirror's Edge: Catalyst concept

Quality challenges

- Getting the last 5-10% quality can be very expensive
  - While covering a relatively small portion of the screen
  - Example: hair & fur rendering
  - Improving GPUs in some of these areas may not benefit “ordinary” rendering
- How to build truly scalable solutions
  - Example: Rendering, lighting and shadowing a full forest
  - Level-of-detail is a key challenge for most techniques to make them practical

Scalable solutions – screen-space

- Sub-surface scattering went from texture- to screen-space
  - Orders of magnitude faster
  - Implicitly scalable + no per-object tracking
  - Not perfect, but made it practical & mainstream

- Volumetric rendering to view frustum 3d texture
  - Froxels! See [Wronski14] and [Hillaire15]

Scalable solutions – screen-space

- Can one extend screen-space techniques further?
- Render multiple depth layers to solve occlusion
  - Multi-layer deep gbuffers [Mara14]
- Render cubemap to reach outside of frustum
  - Render lower resolution separate cubemap, slow
  - Render main view as cubemap with variable resolution?
    - Single geometry pass
    - Also for fovated rendering

Scalable solutions – pre-compute

- Traditionally a strong cut off between pre-computed & runtime solutions

- Believe this is going away more – techniques and systems have to scale & cover more of the spectrum:
  - Offline pre-compute: Highest-quality
  - Load-time pre-compute: High-quality
  - Background compute: Medium-quality
  - Runtime

- Want flexible tradeoffs depending on contexts
  - Artist live editing lighting
  - Gamer customizing in-game content
  - Background gameplay changes to the game environment

Scalable solutions – hierarchical geometry

- Want to avoid wasteful brute force geometry rendering

- Do your own culling, occlusion & LOD directly on the GPU
  - Finer granularity than CPU code
  - Engine can have more context and own spatial data structures
  - Combined with GPU information (for example HiZ)
  - Opportunities to extend the GPU pipeline?

- Compute as frontend for graphics pipeline to accelerate
  - Avoid writing geometry out to memory
  - Good fit with procedural geometry systems as well

Takeaways

- We've gotten very far in the last few years!
  - Big transitions: PBR, Gen4, Compute, explicit APIs
- We are at the cusp of a beautiful future!
- Build your own rendering pipelines & data structures
  - But which ones? All of them! ☺
- Need reduce coupling & further evolve GPU execution models

Thanks to everyone who provided feedback!

- Sébastien Hillaire (@sebhillaire)
- Christina Coffin (@christinacoffin)
- John White (@zedcull)
- Aaron Lefohn (@aaronlefohn)
- Colin Barré-Brisebois (@zigguratvertigo)
- Sébastién Lagarde (@seblagarde)
- Tomasz Stachowiak (@h3r2tic)
- Andrew Lauritzen (@andrewlauritzen)
- Jasper Bekkers (@jasperbekkers)
- Yuiry O'Donnell (@yuriyodonnell)
- Kenneth Brown
- Natasha Tatarchuk (@mirror2mask)
- Angelo Pesce (@kenpex)
- David Reinig (@d13_dreinig)
- Promit Roy (@promit_roy)
- Rich Forster (@dickyjimforster)
- Niklas Nummelin (@niklasnummelin)
- Tobias Berghoff (@tobiasberghoff)
- Morgan McGuire (@casualeffects)
- Tom Forsyth (@tom_forsyth)
- Eric Smolikowski (@esmolikowski)
- Nathan Reed (@reedbeta)
- Christer Ericson (@christerericson)
- Daniel Collin (@daniel_collin)
- Matias Goldberg (@matiasgoldberg)
- Arne Schober (@khipu_kamayuq)
- Dan Olson (@olson_dan)
- Joshua Barczak (@joshuabarczak)
- Bart Wronski (@bartwronsk)
- Krzysztof Narkowicz (@knarkowicz)
- Julien Guertault (@zavie)
- Sander van Rossen (@logicalerror)
- Lucas Hardi (@lhardi)
- Tim Foley (@tangentvector)

Questions?

Email: johan@ea.com
Web: http://frostbite.com
Twitter: @repi

References

- [Andersson07] Terrain rendering in Frostbite using Procedural Shader Splatting
- [Hillaire15] Physically Based and Unified Volumetric Rendering in Frostbite
- [Wronski14] Volumetric fog: Unified, compute shader based solution to atmospheric scattering
- [Salvi15] Anti-Aliasing: Are We There Yet?
- [Mara14] Fast Global Illumination Approximations on Deep G-Buffer
- [Johansson15] Leap of Faith: The World of Mirror’s Edge Catalyst