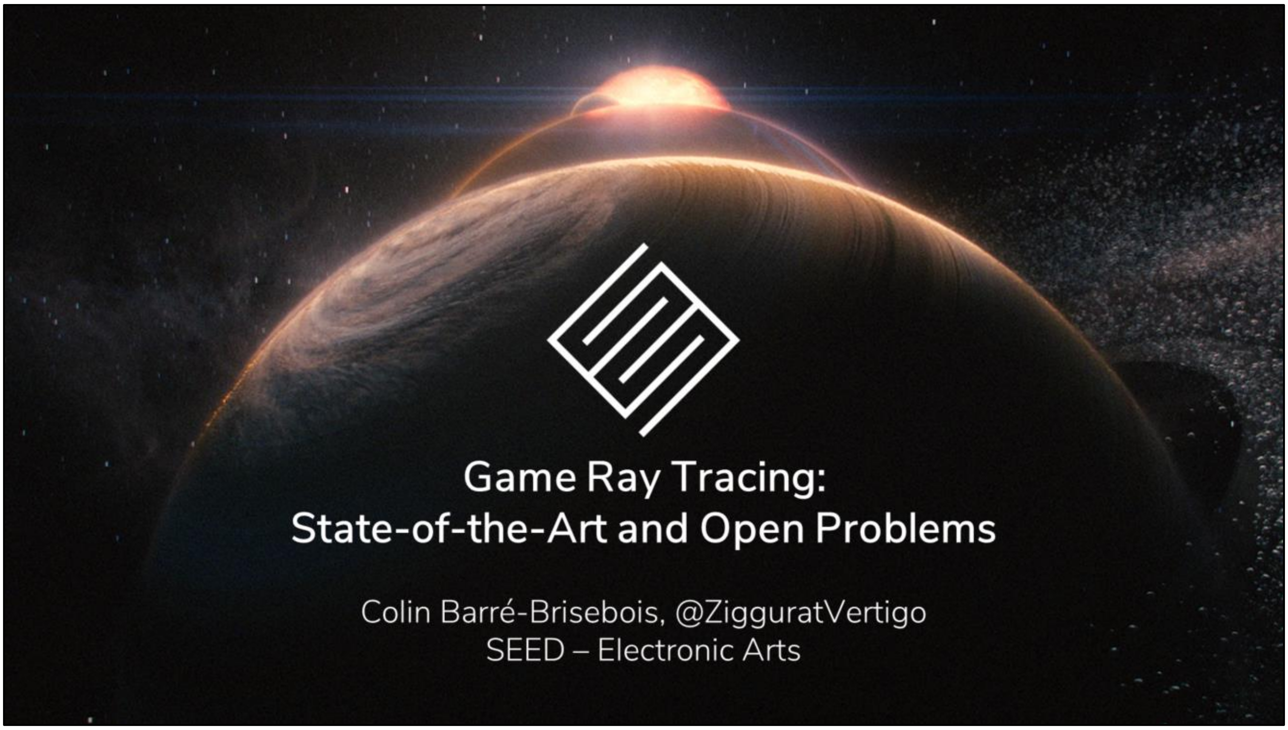
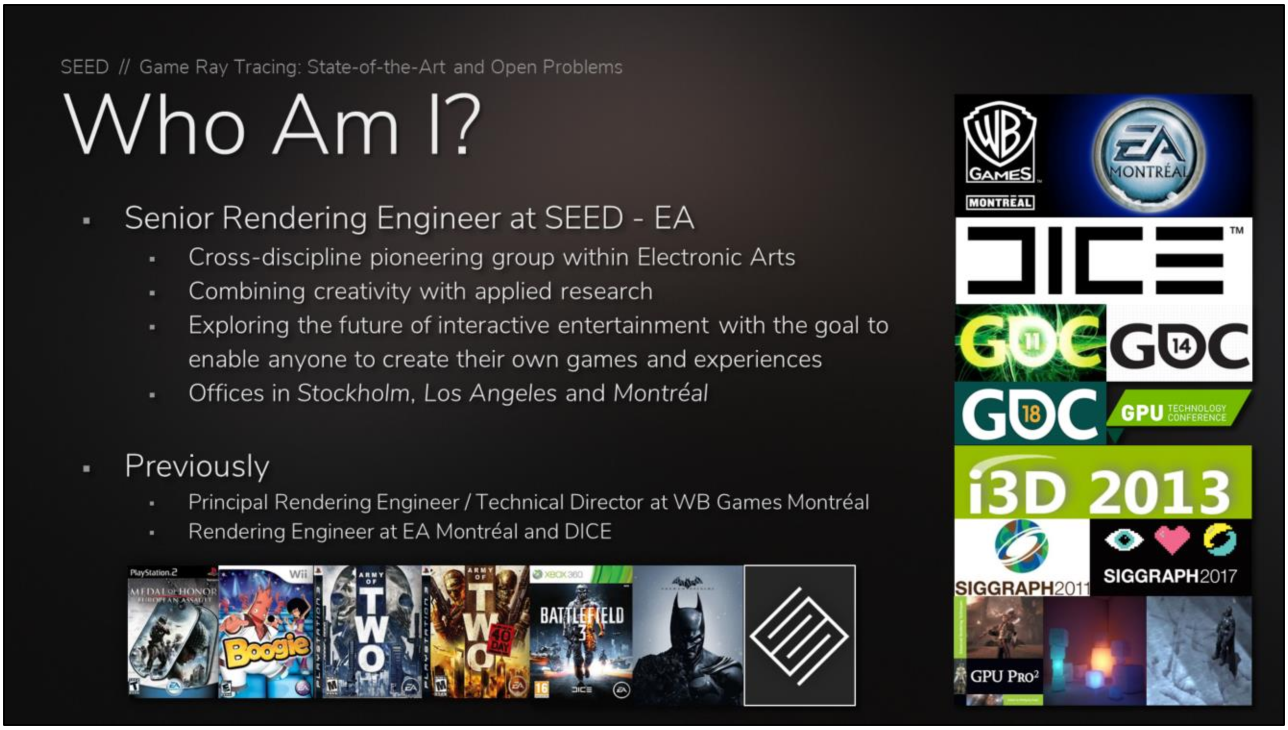


Game Ray Tracing: State-of-the-Art and Open Problems

Colin Barré-Brisebois, @ZigguratVertigo
SEED – Electronic Arts

- Good morning everyone and thanks for making it so early! ☺
- My name is Colin Barré-Brisebois and today I would like to talk to you about some of the state of the art in real-time ray tracing for games
- I also want to discuss some open problems several of us in the games industry see, that's well need some help with and hopefully tackle together with you

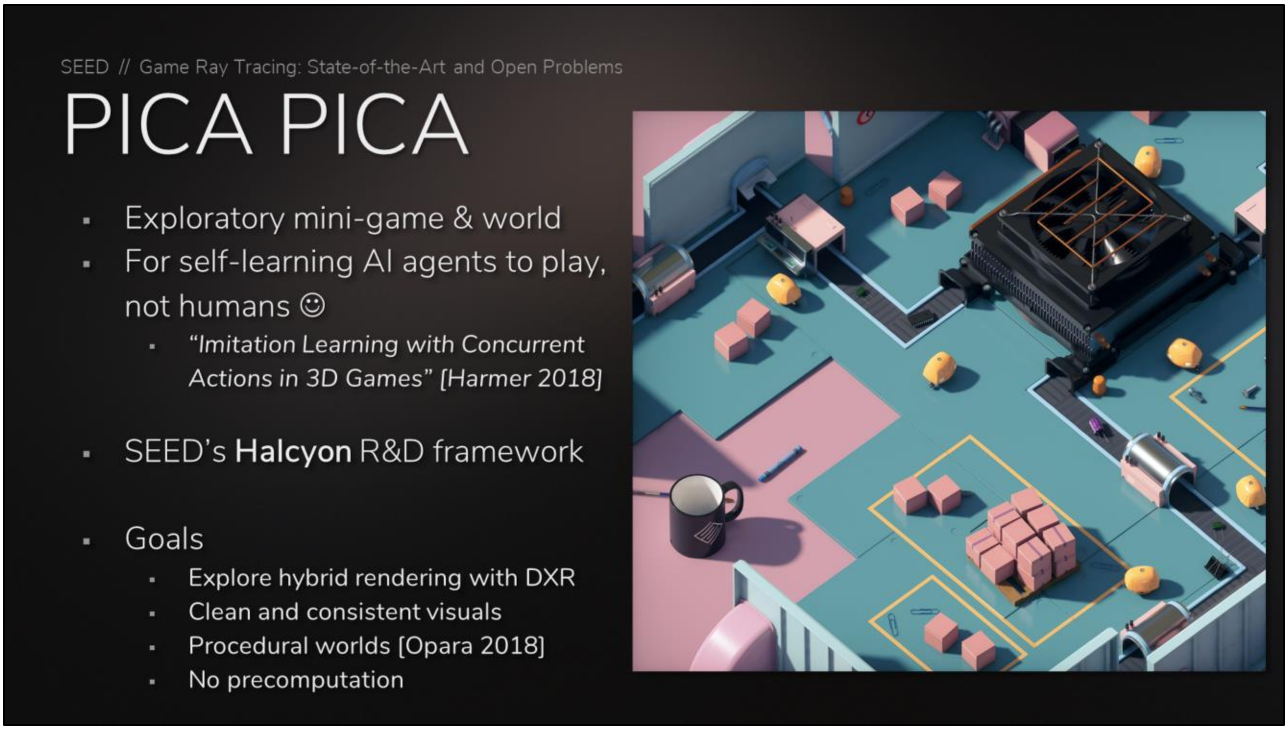


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Who Am I?

- Senior Rendering Engineer at SEED - EA
  - Cross-discipline pioneering group within Electronic Arts
  - Combining creativity with applied research
  - Exploring the future of interactive entertainment with the goal to enable anyone to create their own games and experiences
  - Offices in *Stockholm*, *Los Angeles* and *Montréal*
- Previously
  - Principal Rendering Engineer / Technical Director at WB Games Montréal
  - Rendering Engineer at EA Montréal and DICE

- Who is SEED? For those of you who don't know, we're a technical and creative research division inside Electronic Arts.
- We are a cross-disciplinary team whose mission to explore the future of interactive entertainment, with the goal to enable anyone to create their own games and experiences.
- We have offices in Stockholm, Los Angeles and Montréal.
- One of our recent projects was an experiment with hybrid real-time rendering, deep learning agents, and procedural level generation.
- We presented this at this year's GDC. In case you haven't see it, let's check it out!

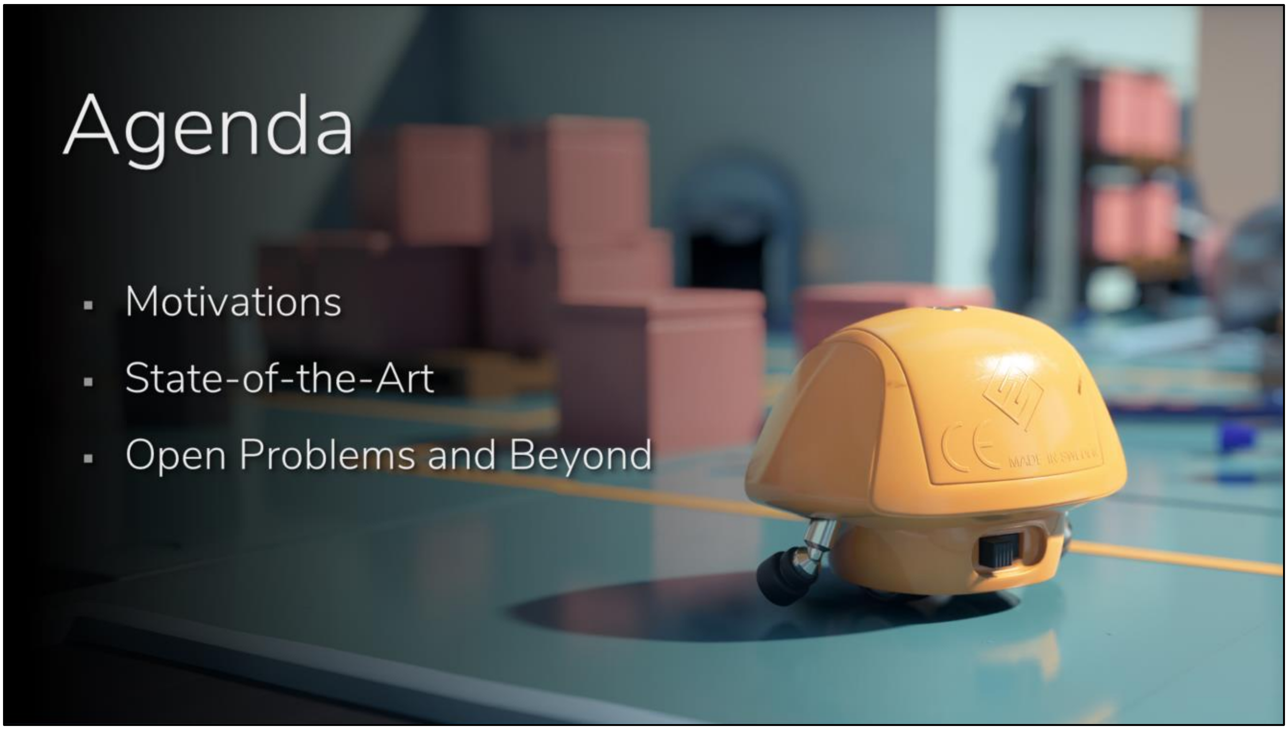


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

PICA PICA

- Exploratory mini-game & world
- For self-learning AI agents to play, not humans ☺
  - *"Imitation Learning with Concurrent Actions in 3D Games" [Harmer 2018]*
- SEED's **Halcyon** R&D framework
- Goals
  - Explore hybrid rendering with DXR
  - Clean and consistent visuals
  - Procedural worlds [Opara 2018]
  - No precomputation

- PICA PICA is a mini-game that we built for AI agents rather than humans.
- Using reinforcement learning, the agents learn to navigate and interact with the environment. They run around and fix the various machines, so that conveyor belts keep running efficiently. If you are interested in the AI part, do check out our paper in arXiv.
- We built the mini-game from the ground up in our in-house Halcyon R&D framework.
- We've had the opportunity to be involved early on with DirectX Ray Tracing, with NVIDIA and Microsoft, to explore some of the possibilities with this technology.
- We decided to create something a bit different and unusual, less AAA than what you usually expect from an EA title.
- For this demo we wanted cute visuals that would be clean and stylized, yet grounded in physically-based rendering. We wanted to showcase the strengths of ray tracing, while also taking into account our small art department of 2 people.
- We used procedural level generation with an algorithm that drove layout and asset placement. You should check out Anastasia's various talks on the matter in case you want to know more how the world was procedurally generated.



Agenda

- Motivations
- State-of-the-Art
- Open Problems and Beyond

Why use ray tracing?

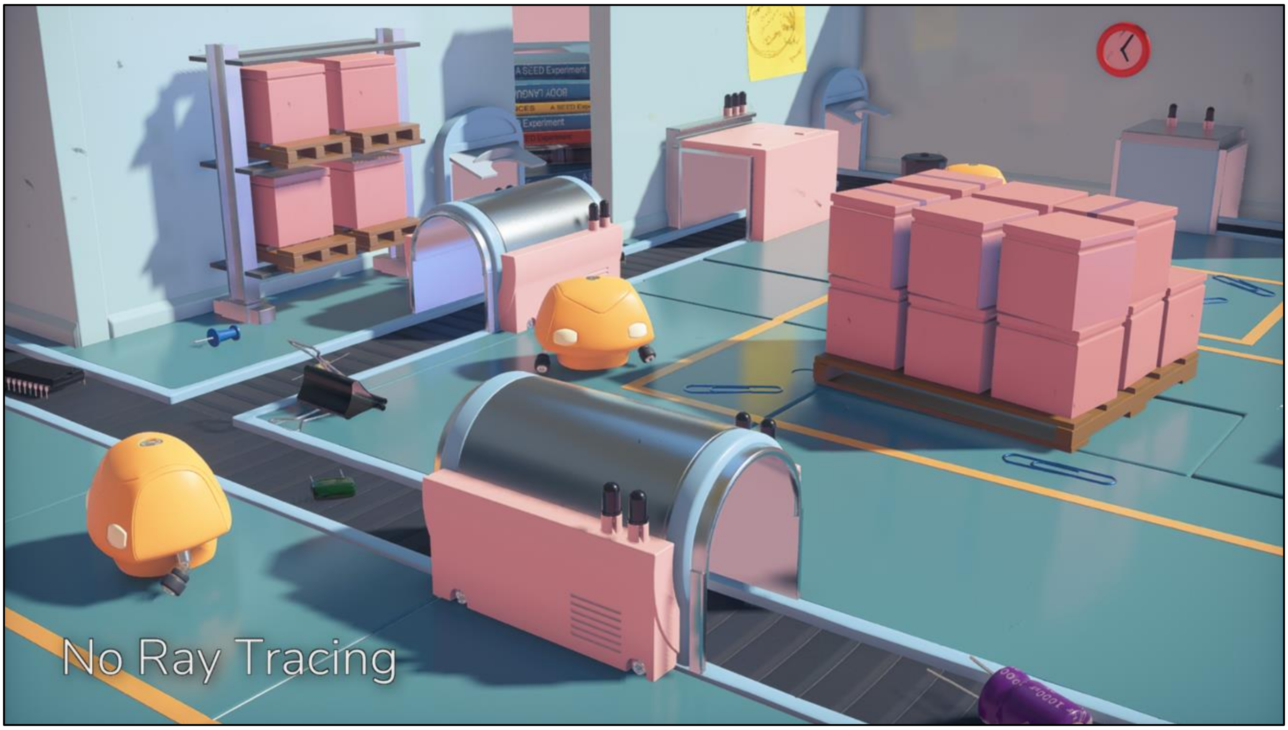
So here’s a question: why should we use ray tracing?

I guess for most folks in this room the questions doesn’t need to be asked.

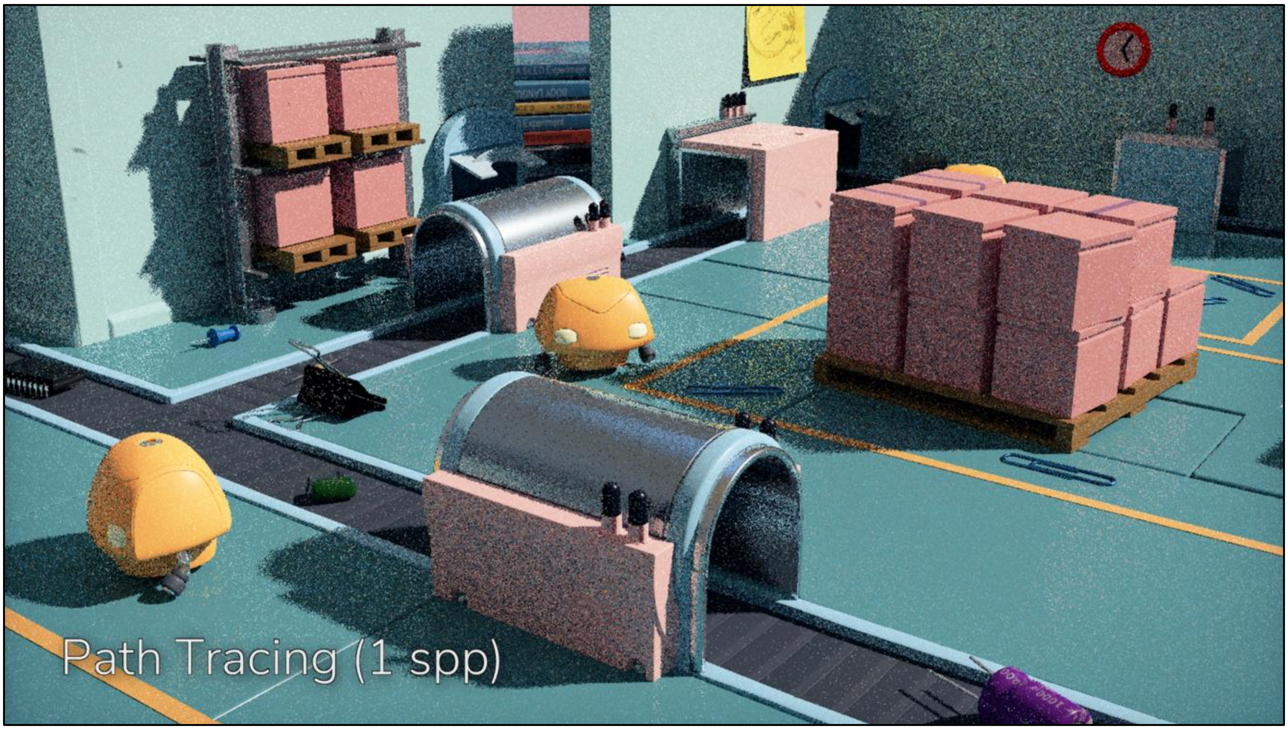
And the best answer is…

Why not!

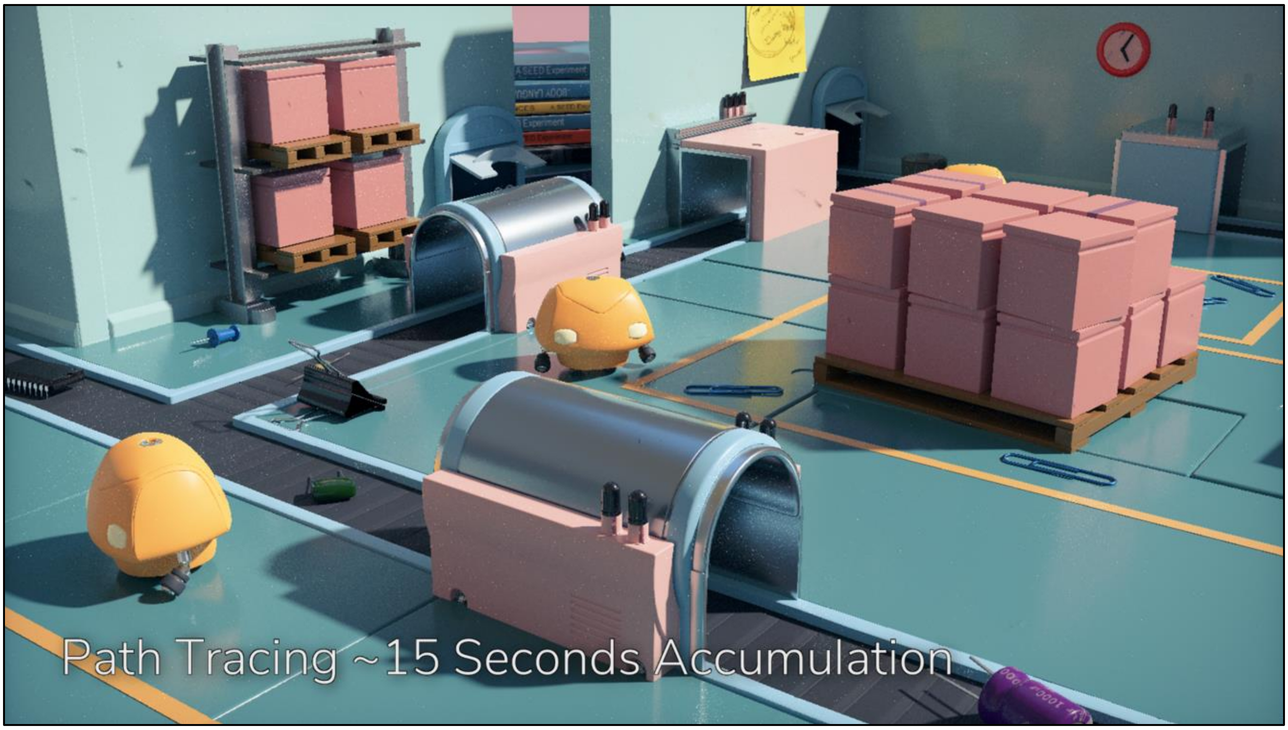
- Because why not! And because we can!!
- We can take a stab at “ray tracing is the future and ever will be”.
- Let’s put things into perspective.



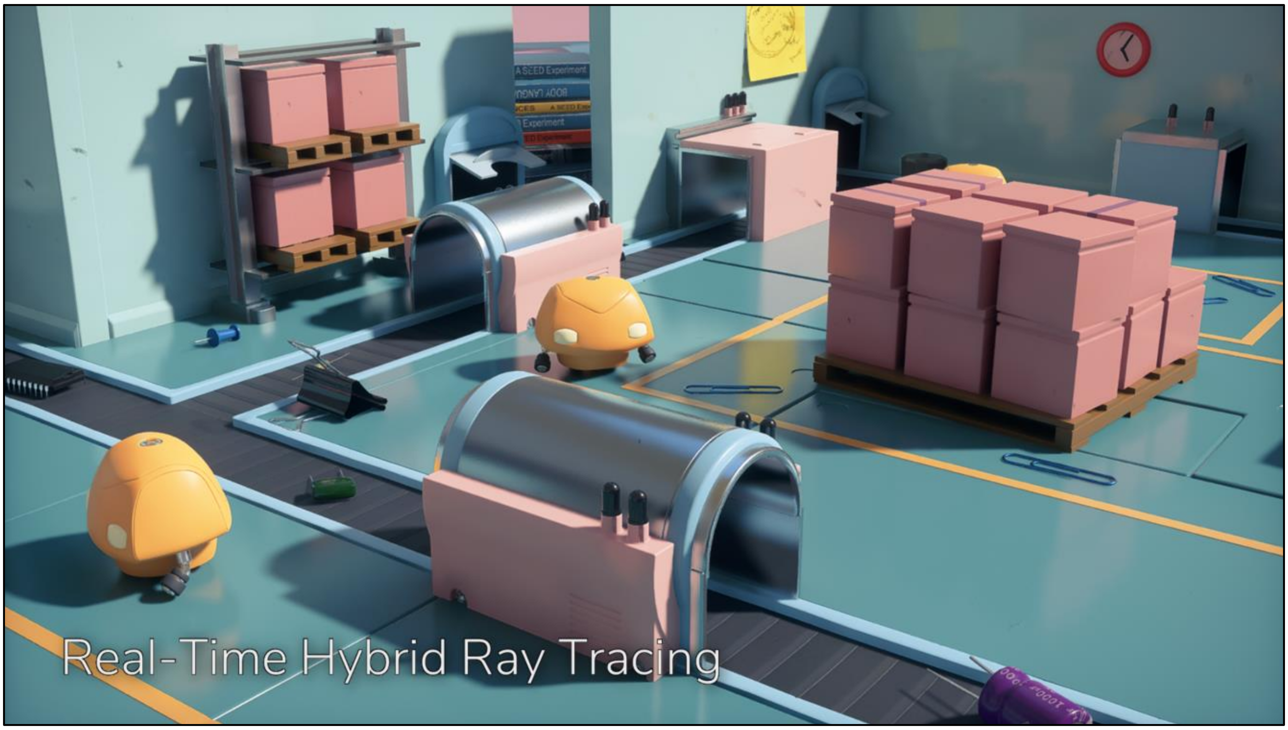
- This is our original scene without ray tracing. It’s not super awesome.
- It does include a number of non-trivial rendering algorithms such as screen-space reflections, screen-space ambient occlusion, physically-based rendering, cascaded shadow maps, a multi-layer PBR material system, a post-fx stack with eye adaptation, motion blur, depth of field, and temporal-antialiasing.



If we turn our in-engine path tracer on, at one sample-per-pixel, the scene looks considerably noisy and is rather expensive to render.

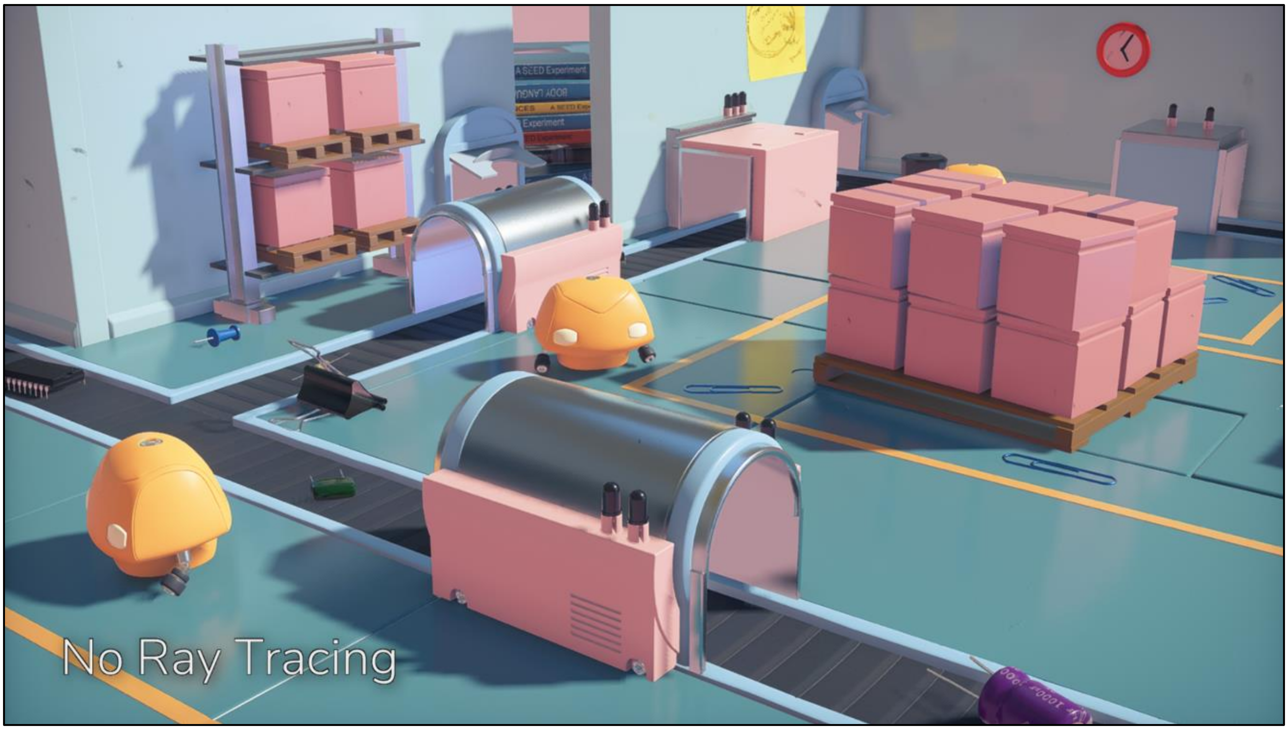


- After waiting for 15 seconds, this what we get.
- The image looks way nicer but you will also notice some noise.
- Some of that noise can take minutes to hours to converge, mostly because of difficult light paths and caustics.

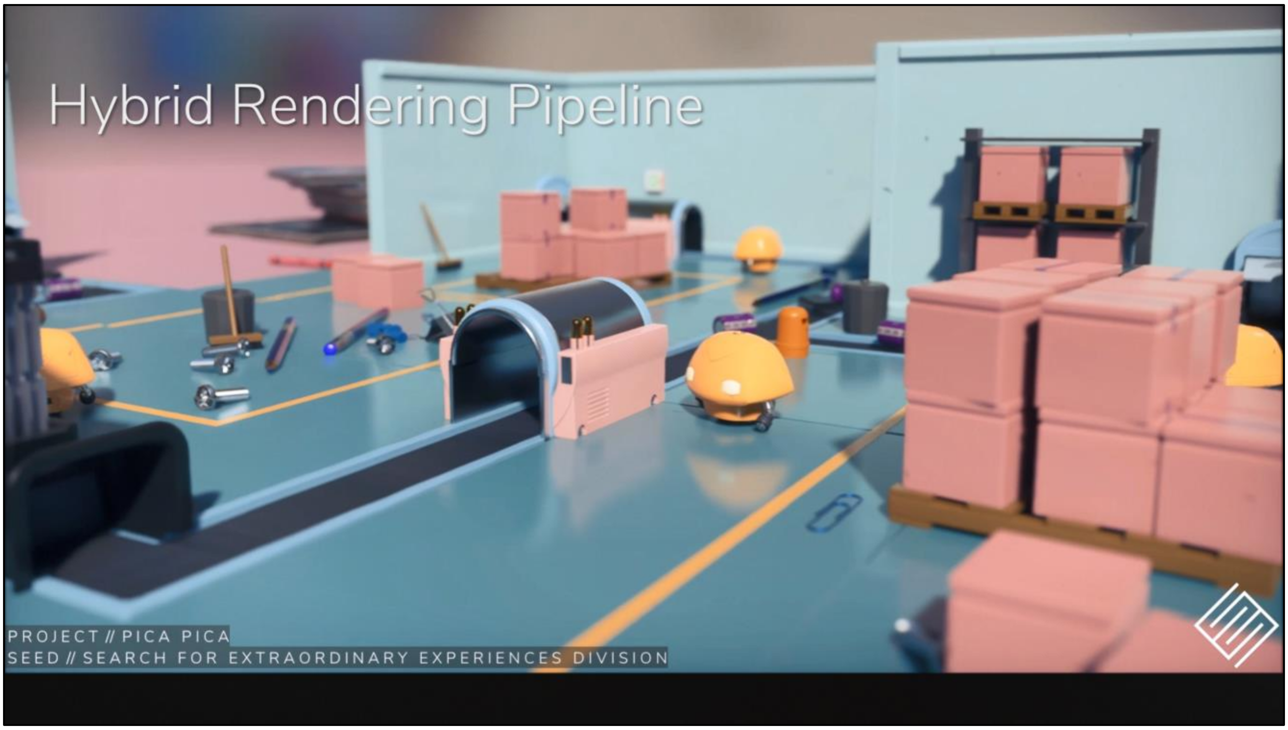


Real-Time Hybrid Ray Tracing

- But instead if we adopt an hybrid approach, this is the kind of results we get running at 60 FPS on a TitanV.
- It’s not quite the same as the path traced version; it’s missing caustics and some small-scale interreflections but still pretty decent.



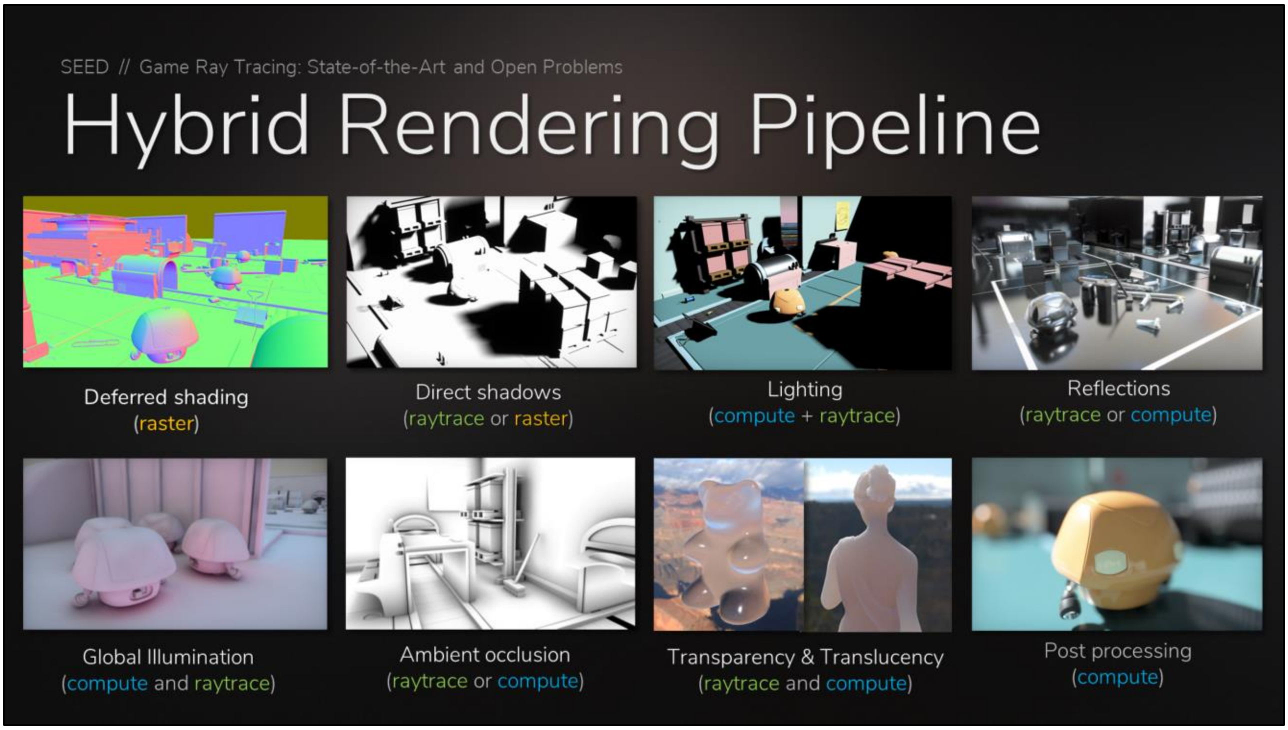
So yeah, no ray tracing. And with ray tracing (flip)



Hybrid Rendering Pipeline

PROJECT // PICA PICA
SEED // SEARCH FOR EXTRAORDINARY EXPERIENCES DIVISION

- For PICA PICA we built a hybrid rendering pipeline which uses both ray tracing, compute and rasterization.
- By using the interoperability of DirectX, we are able to use the best of all pipeline stages and achieve a final image augmented by ray tracing at real-time rates, good enough to be in a game product.
- In the end we get a super stable image that's noticeably free of noise, noise which you usually get from monte carlo while benefiting from what you get from ray tracing, such as awesome reflections, AO, GI, and respecting physically-based rendering.



SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Hybrid Rendering Pipeline

Deferred shading
(raster)

Direct shadows
(raytrace or raster)

Lighting
(compute + raytrace)

Reflections
(raytrace or compute)

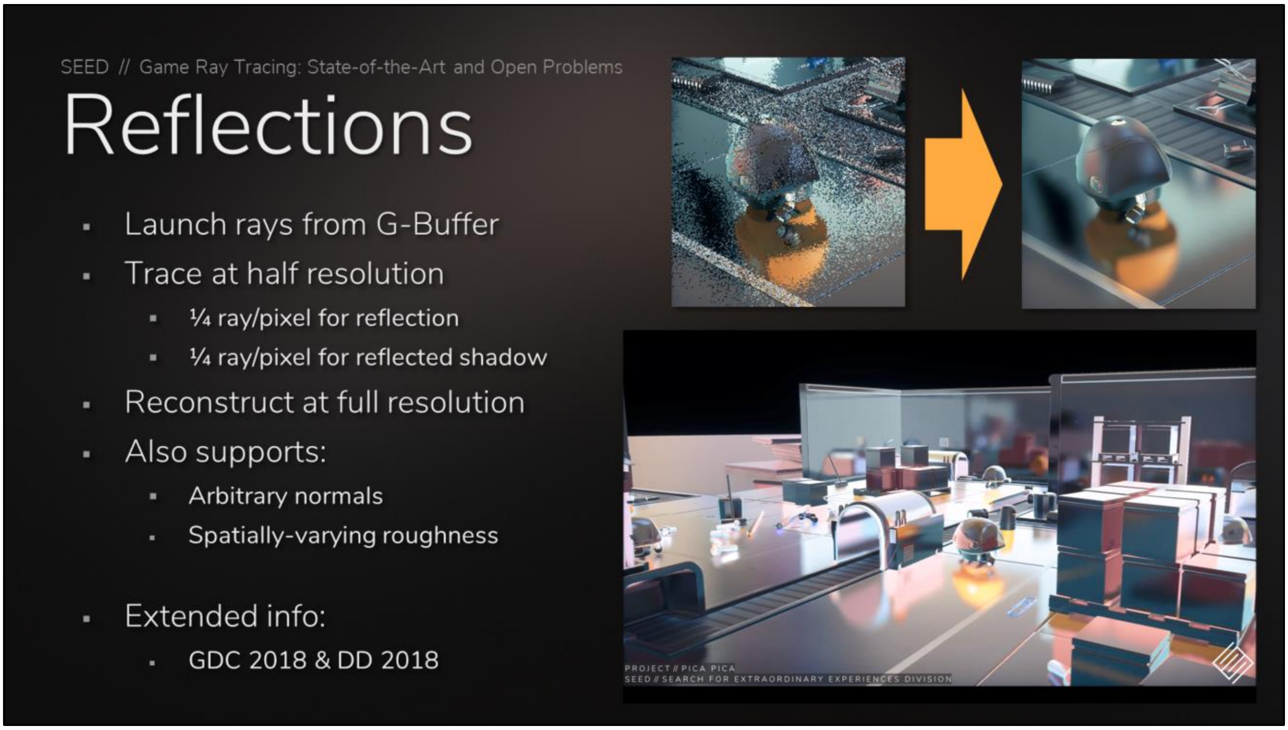
Global Illumination
(compute and raytrace)

Ambient occlusion
(raytrace or compute)

Transparency & Translucency
(raytrace and compute)

Post processing
(compute)

- This is what our hybrid rendering pipeline looks like.
- As mentioned we take advantage of the interop between rasterization, compute and ray tracing, which you can see with the various effects here.
- We have a standard deferred renderer with compute-based lighting, and a pretty standard post-processing stack.
- We can render shadows via ray tracing or cascaded shadow maps.
- Reflections can be raytraced or screen-space raymarched. Same story for ambient occlusion.
- Only global illumination, transparency and translucency fully require ray tracing.

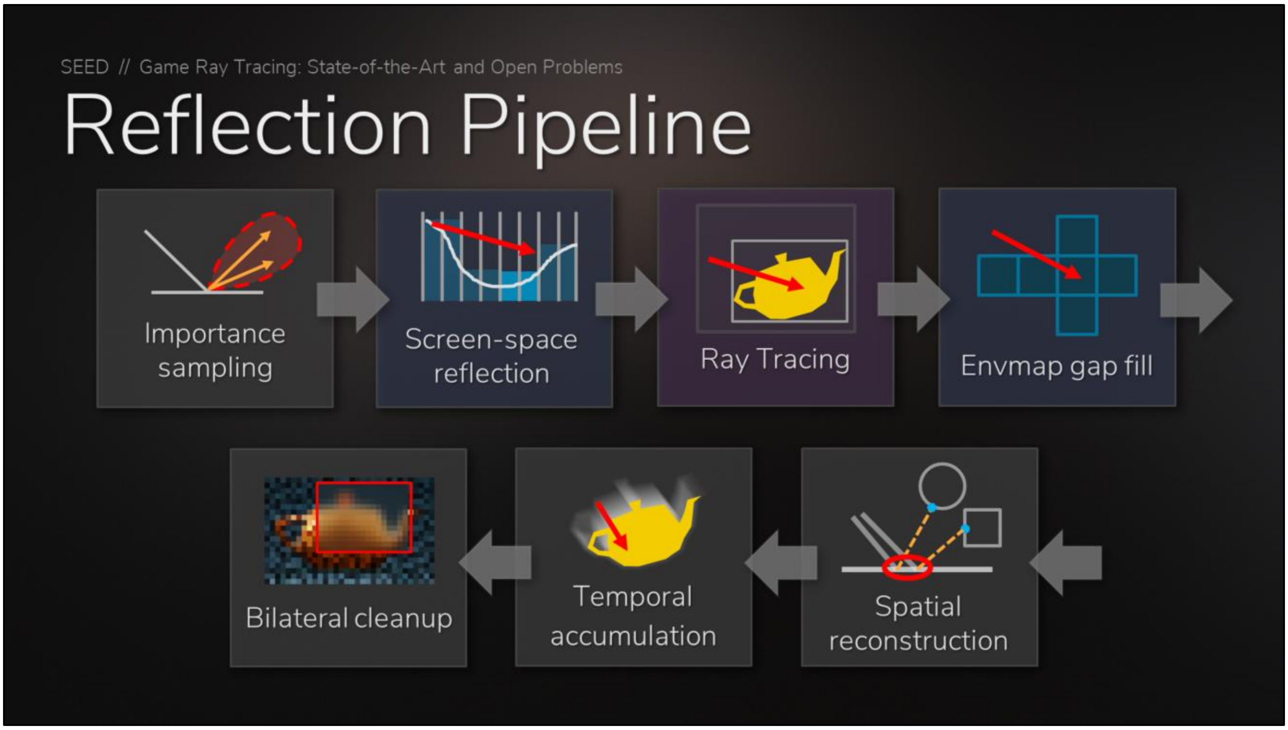


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Reflections

- Launch rays from G-Buffer
- Trace at half resolution
  - ¼ ray/pixel for reflection
  - ¼ ray/pixel for reflected shadow
- Reconstruct at full resolution
- Also supports:
  - Arbitrary normals
  - Spatially-varying roughness
- Extended info:
  - GDC 2018 & DD 2018

- One of our main techniques which takes advantage of ray tracing is of course reflections
- The ability to launch rays from the g-buffer allows us to trace reflections faster than through primary visibility.
- We trace them at half resolution, so for every four pixels you get one reflection ray. This gives us a quarter of a ray per pixel.
- Then at the hit point, shadows will typically be sampled with another ray. This totals to half a ray per pixel.
- This works for hard reflections, but we also support arbitrary normal and varying roughness.
- We do this by applying spatiotemporal reconstruction and filtering
- Our first approach combined it with SSR for performance, but in the end we just raytraced for simplicity and uniformity.
- We have presented some extended info on how we do this, so check out our GDC 2018 and Digital Dragons talks



SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Reflection Pipeline

Importance sampling

Screen-space reflection

Ray Tracing

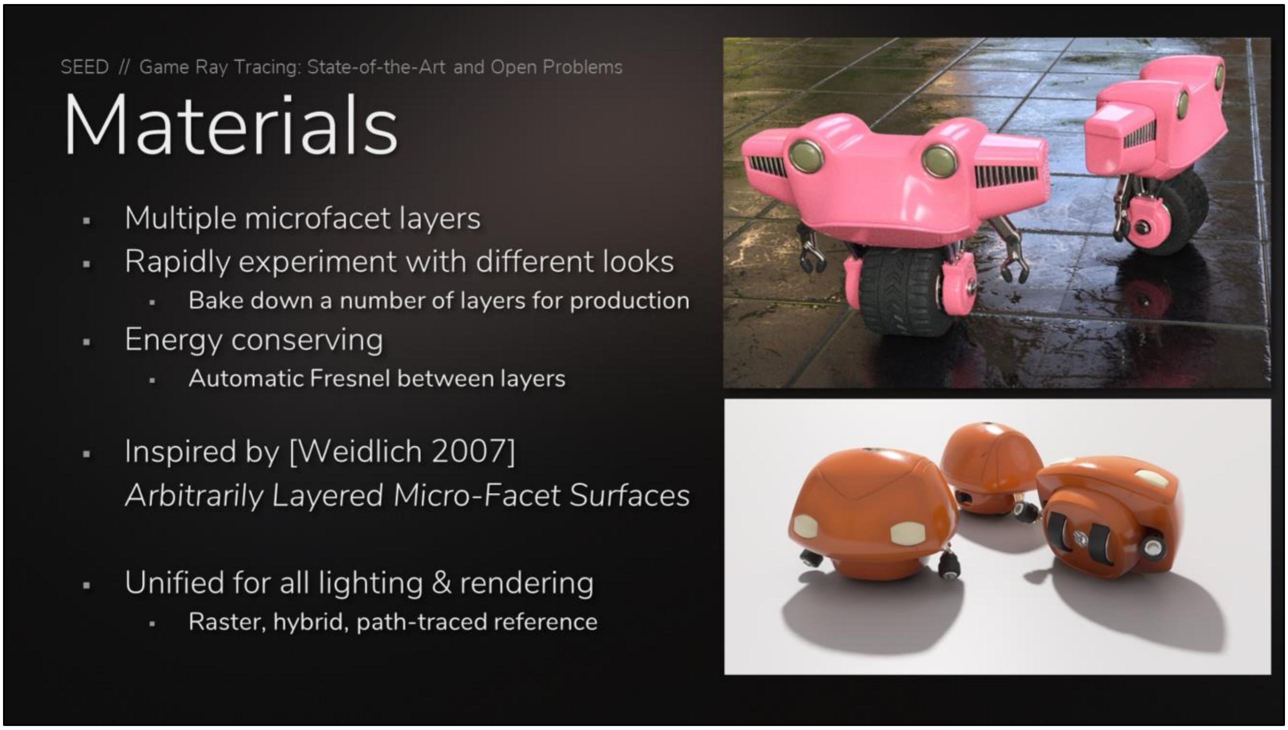
Envmap gap fill

Spatial reconstruction

Temporal accumulation

Bilateral cleanup

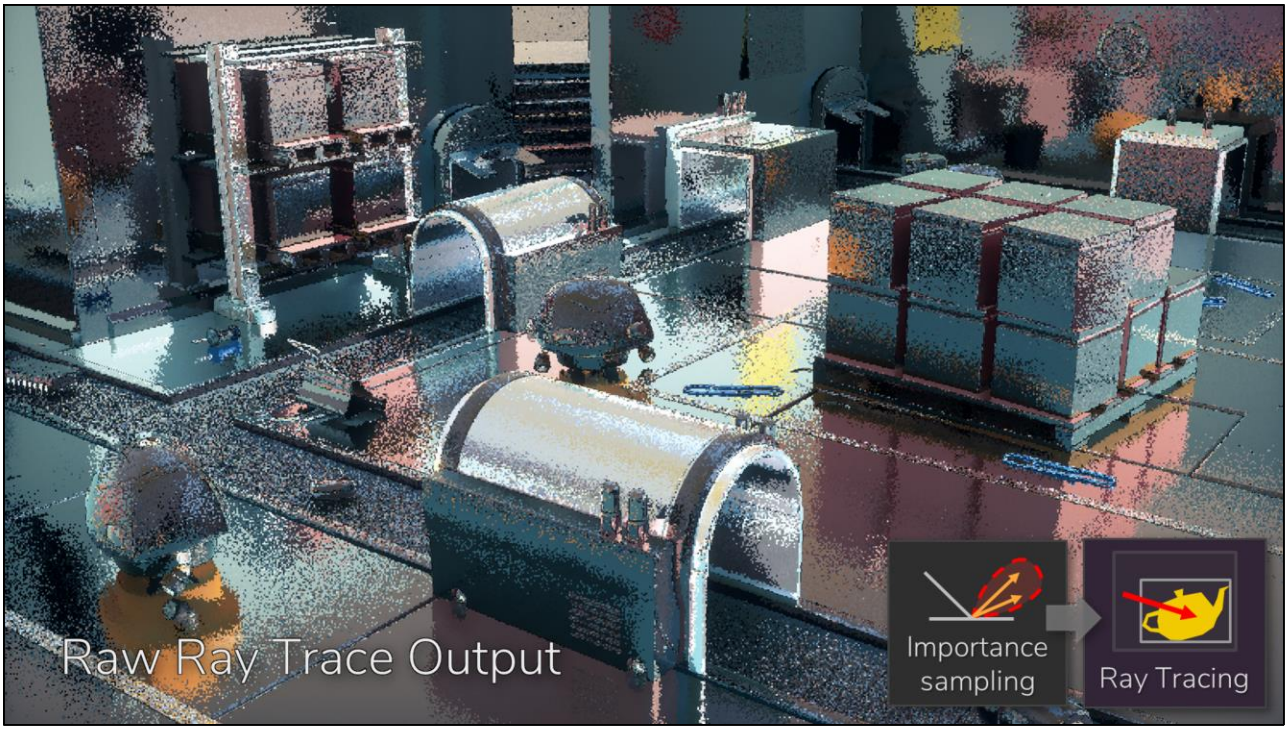
- Here’s a high-level summary of the ray tracing pipeline.
- We start by generating rays via BRDF importance sampling. This gives us rays that follow the properties of our materials.
- Scene intersection can then be done either by screen-space raymarching or ray tracing. As I said in the end we only raytrace but support both.
- Once intersections have been found, we proceed to reconstruct the reflected image.
- Our kernel reuses ray hit information across pixels, upsampling the image to full-resolution.
- It also calculates useful information for a temporal accumulation pass.
- Finally we run a last-chance noise cleanup in the form of a cross-bilateral filter.



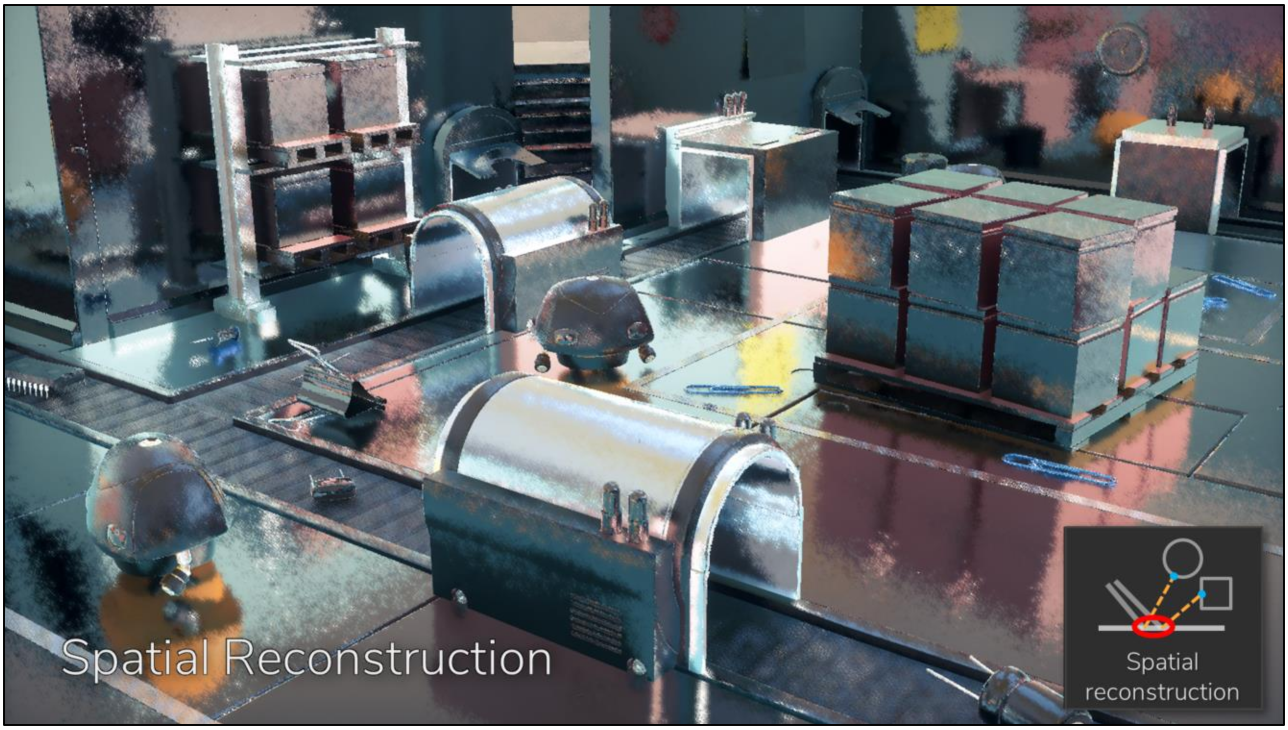
Materials

- Multiple microfacet layers
- Rapidly experiment with different looks
  - Bake down a number of layers for production
- Energy conserving
  - Automatic Fresnel between layers
- Inspired by [Weidlich 2007]
  Arbitrarily Layered Micro-Facet Surfaces
- Unified for all lighting & rendering
  - Raster, hybrid, path-traced reference

- When it comes to materials, for this project we put together a material model where we combine multiple layers into a single, unified and expressive BRDF.
- By unified I mean that this model works for all lighting & rendering modes, is energy conserving and handles Fresnel between layers.
- It has allowed us to rapidly experiment with different looks, but if we were to use this in a real game production we'd most likely bake down the number of layers.
- The reflection pass works with layered materials, and we still generate just one direction for the whole stack.
- The simplest way to do this is by choosing one of the layers with uniform probability, and sampling the layer's BRDF. That works, but can be wasteful.
- For example, a smooth clear coat layer is barely visible heads-on, but noticeable at grazing angles. To improve on the scheme, we draw the layer from a probability mass function based on each layer's approximate visibility.



- Looking only at the reflections, this is the raw results we get at 1 reflection ray every 4 pixels.

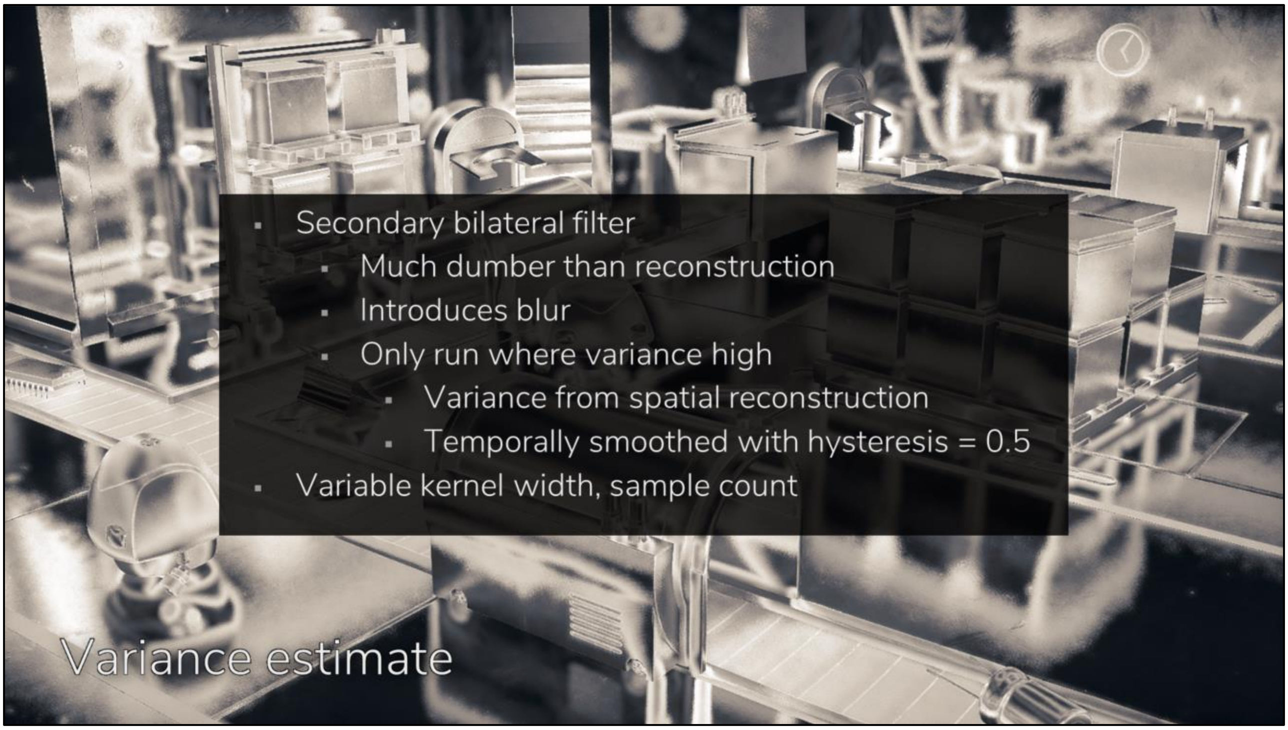


- And this is what the spatial filter does with it. The output is still noisy, but it is now full rez, and it gives us variance reduction similar to actually shooting 16 rays per pixel.
- The idea is very similar to stochastic screen-space reflections that Tomasz Stachowiak from our team presented three years ago at SIGGRAPH. If you want more info about this you should check his talk.
- Every full resolution pixel basically uses a set of ray hits to reconstruct its reflection. It’s a fancy weighted average where the local pixel’s BRDF is used to weigh contributions. It’s also scaled by the inverse PDF of the source rays to account for their distribution.

+Temporal Accumulation

Temporal accumulation

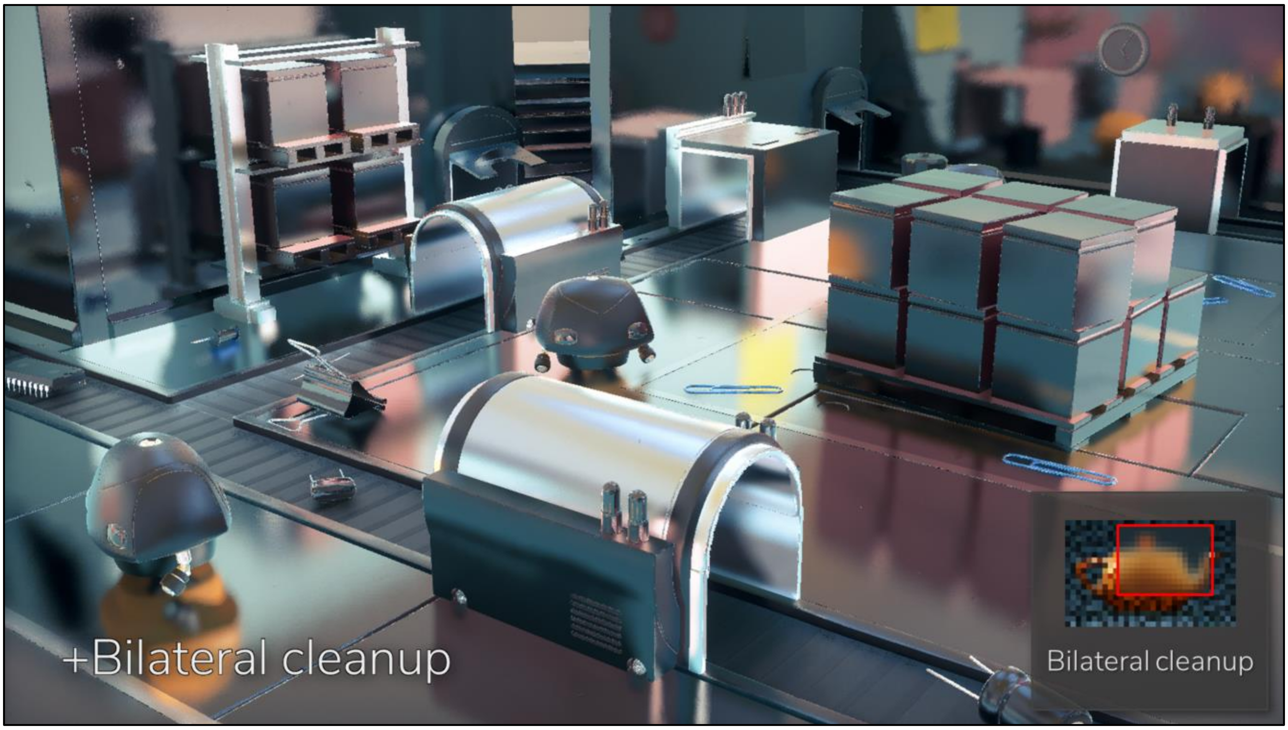
- Followed by temporal accumulation



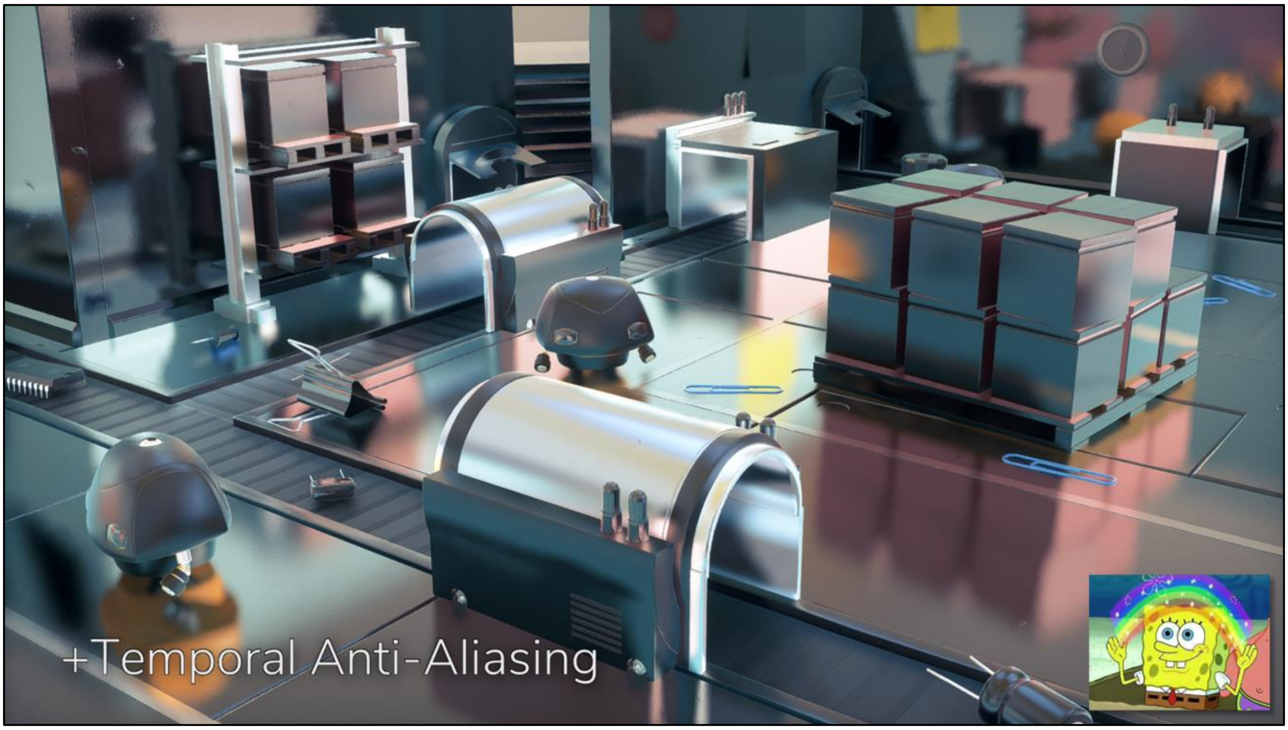
- Secondary bilateral filter
  - Much dumber than reconstruction
  - Introduces blur
  - Only run where variance high
    - Variance from spatial reconstruction
    - Temporally smoothed with hysteresis = 0.5
- Variable kernel width, sample count

Variance estimate

- And finally by a much simpler bilateral filter that removes up some of the remaining noise.
- It overblurs a bit, but we needed it for some of the rougher reflections.
- Compared to SSR, ray tracing is trickier because we can't cheat with a blurred version of the screen for pre-filtered radiance
- There's much more noise compared to SSR, so our filters need to be more aggressive too.
- To prevent it from overblurring, we use the variance estimate from the spatial reconstruction pass and use it to scale down the bilateral kernel size and sample count

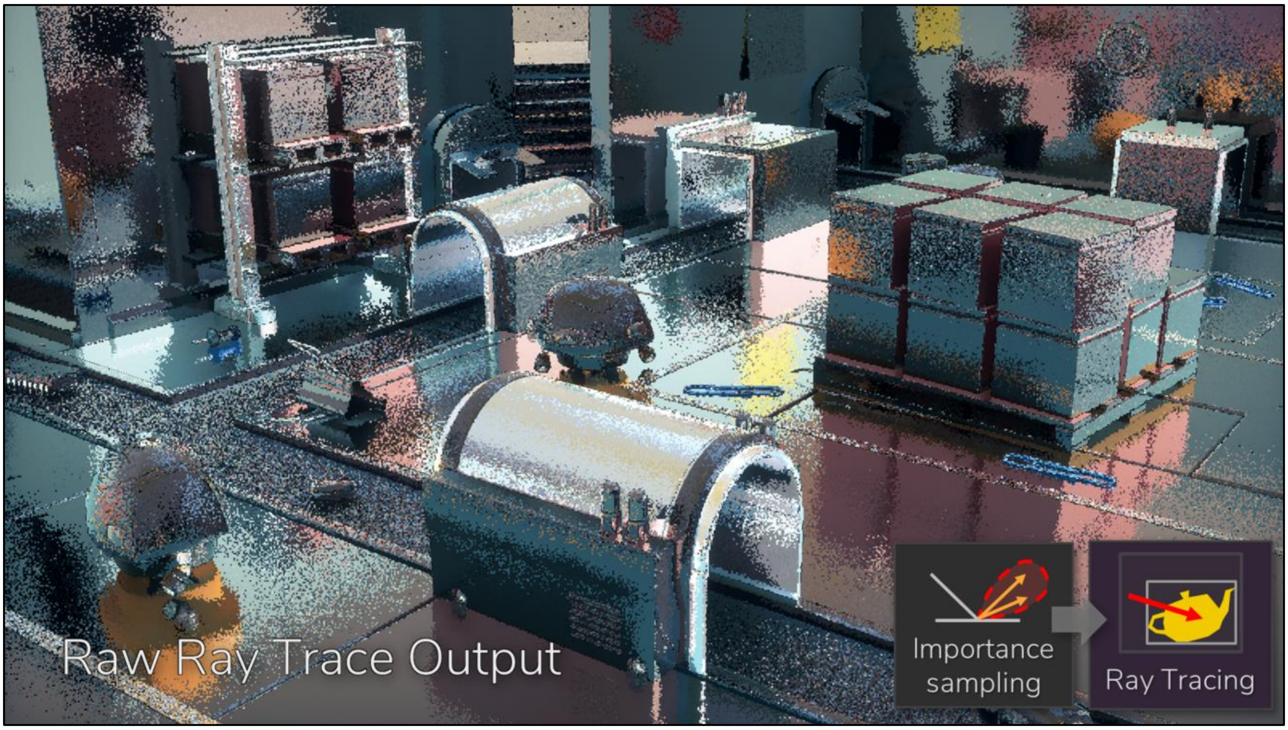


- And finally by a much simpler bilateral filter that removes up some of the remaining noise.
- It overblurs a bit, but we needed it for some of the rougher reflections.
- Compared to SSR, ray tracing is trickier because we can't cheat with a blurred version of the screen for pre-filtered radiance
- There's much more noise compared to SSR, so our filters need to be more aggressive too.
- To prevent it from overblurring, we use the variance estimate from the spatial reconstruction pass and use it to scale down the bilateral kernel size and sample count

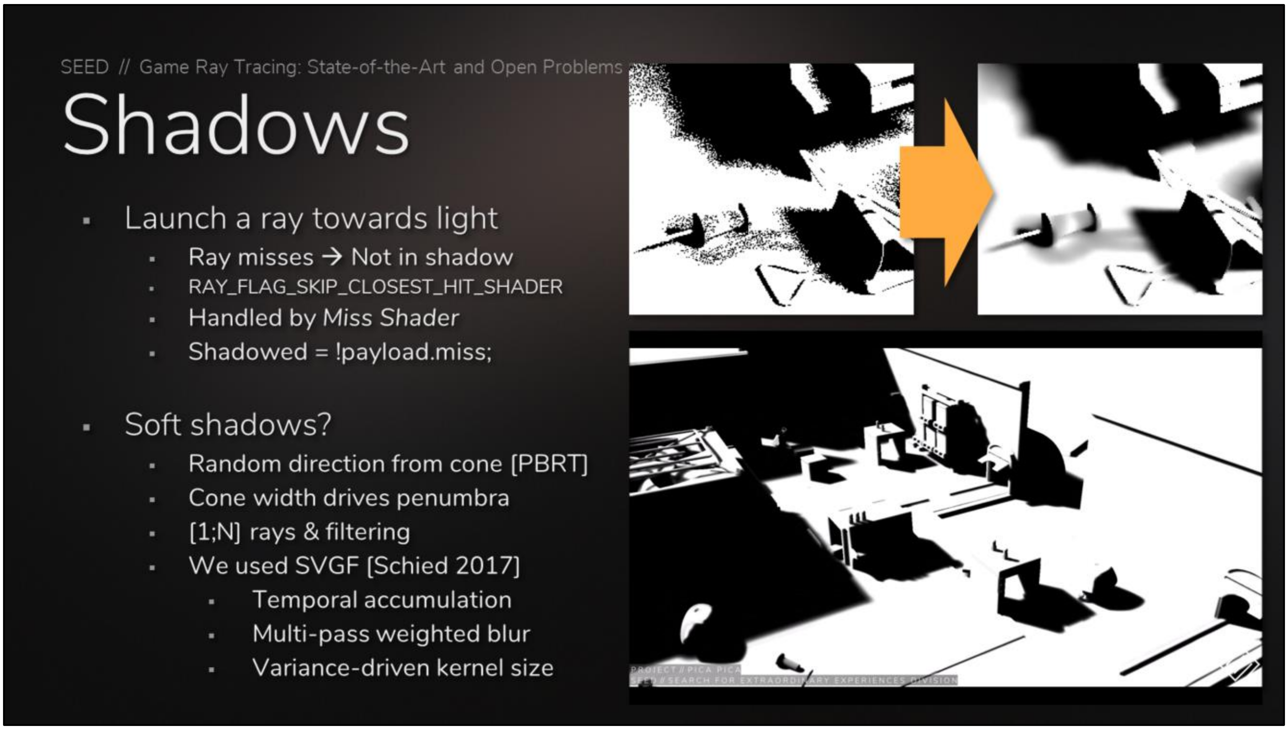


+Temporal Anti-Aliasing

- With temporal anti-aliasing, we then get a pretty clean image. Considering this comes from one quarter rays per pixel per frame, and works with dynamic camera and object movement, it’s quite awesome what can be done when reusing spatial and temporal



- Going back to the raw output, it’s quite a change!

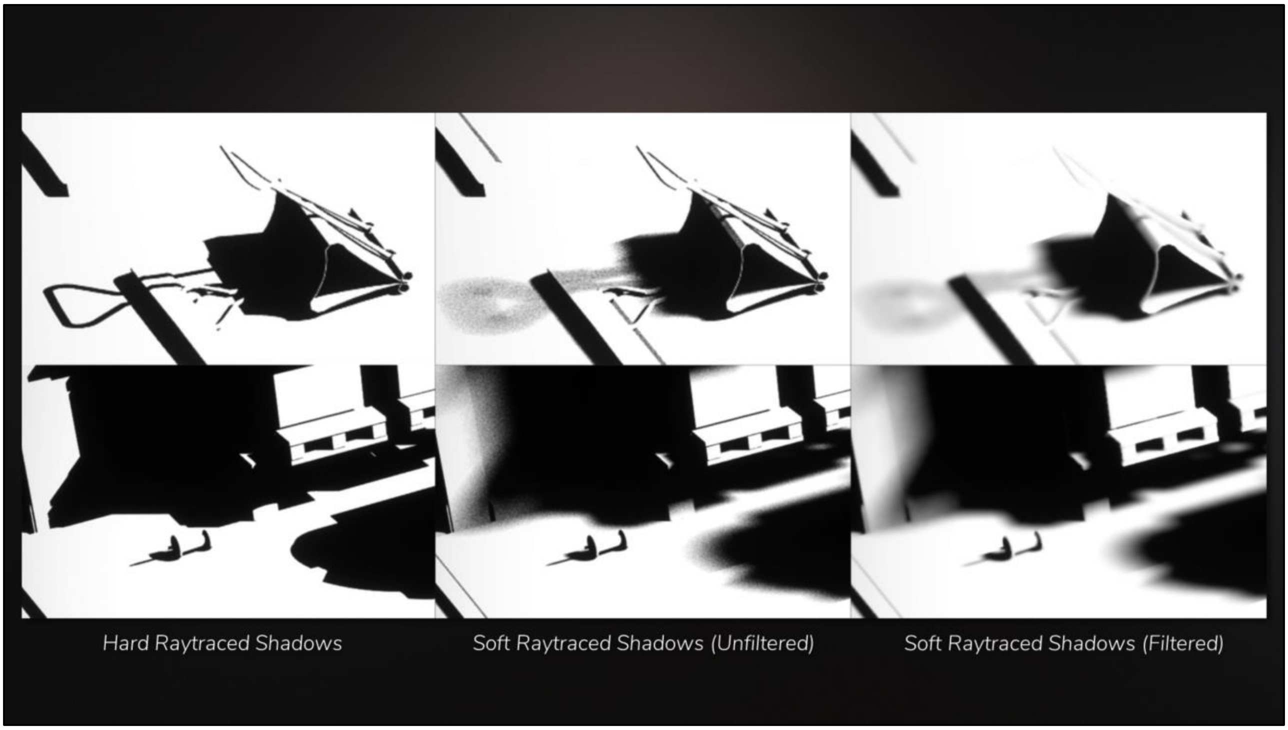


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Shadows

- Launch a ray towards light
  - Ray misses → Not in shadow
  - RAY_FLAG_SKIP_CLOSEST_HIT_SHADER
  - Handled by *Miss Shader*
  - Shadowed = !payload.miss;
- Soft shadows?
  - Random direction from cone [PBRT]
  - Cone width drives penumbra
  - [1;N] rays & filtering
  - We used SVGF [Schied 2017]
    - Temporal accumulation
    - Multi-pass weighted blur
    - Variance-driven kernel size

- Raytraced shadows is another technique we have. Those are great because they perfectly ground objects in the scene. This is not too complicated to implement. Just launch a ray towards the light, and if the ray misses you're not in shadow
- Hard shadows are great… but soft shadows are definitely better to convey scale and more representative of the real world
- This can be implemented by sampling random directions in a cone towards the light
- The wider the cone angle, the softer shadows get but the more noise you'll get, so we have to filter it
- You can launch more than one ray, but will still require some filtering
- In our case, we first reproject results TAA-style, and accumulate shadow and variance
- We then apply an SVGF-like filter. We modified SVGF to make it more responsive for shadows. We coupled it with a color bounding box clamp similar to the one in temporal-antialiasing, with a single scalar value which is cheap to evaluate. We used Marco Salvi's variance-based bounding box method, with a 5x5 kernel estimating the standard deviation in the new frame. The result is not perfect, but the artifacts aren't noticeable with full shading.

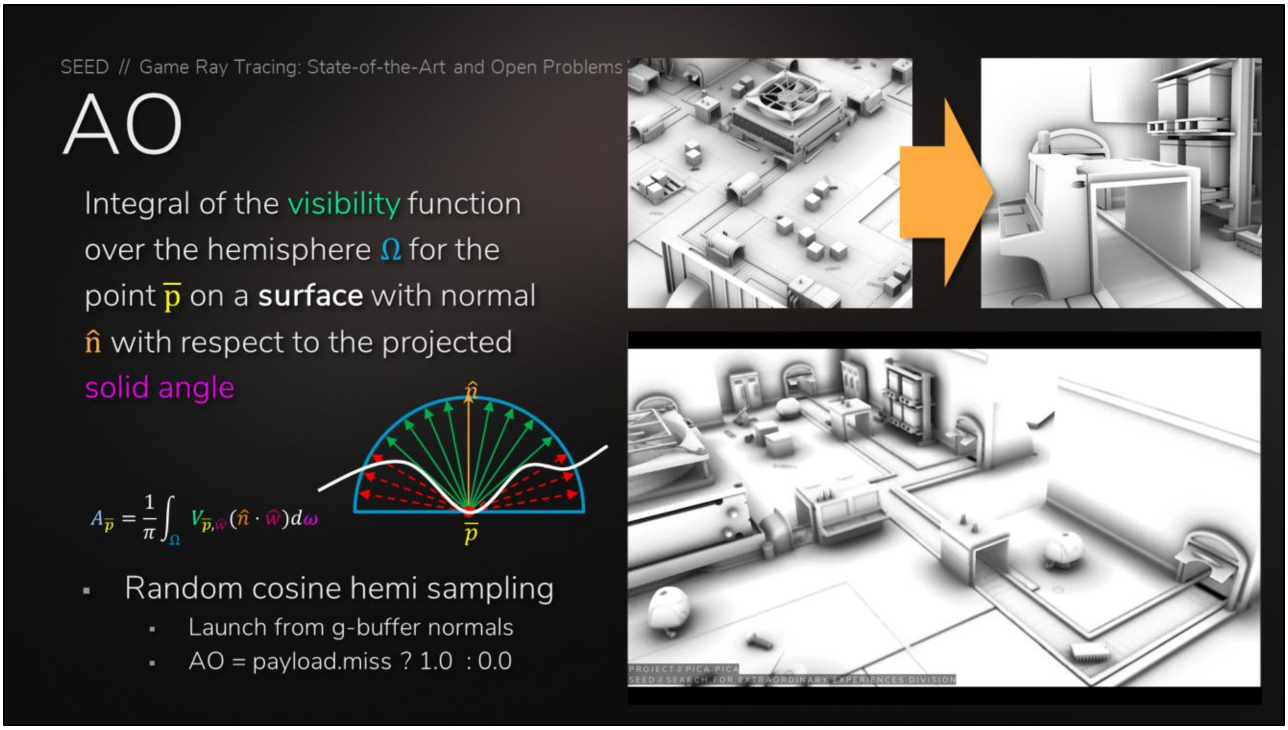


Hard Raytraced Shadows

Soft Raytraced Shadows (Unfiltered)

Soft Raytraced Shadows (Filtered)

- Let’s zoom on some details. With our approach one can see we get nice contact hardening while getting rid of the noise.
- But this approach is not completely physically correct, since we don’t incorporate the BRDF.
- Steve Hill will talk about how you can do this properly in the Advances in Real-Time Rendering Course on Monday at SIGGRAPH. So make sure to attend!



SEED // Game Ray Tracing: State-of-the-Art and Open Problems

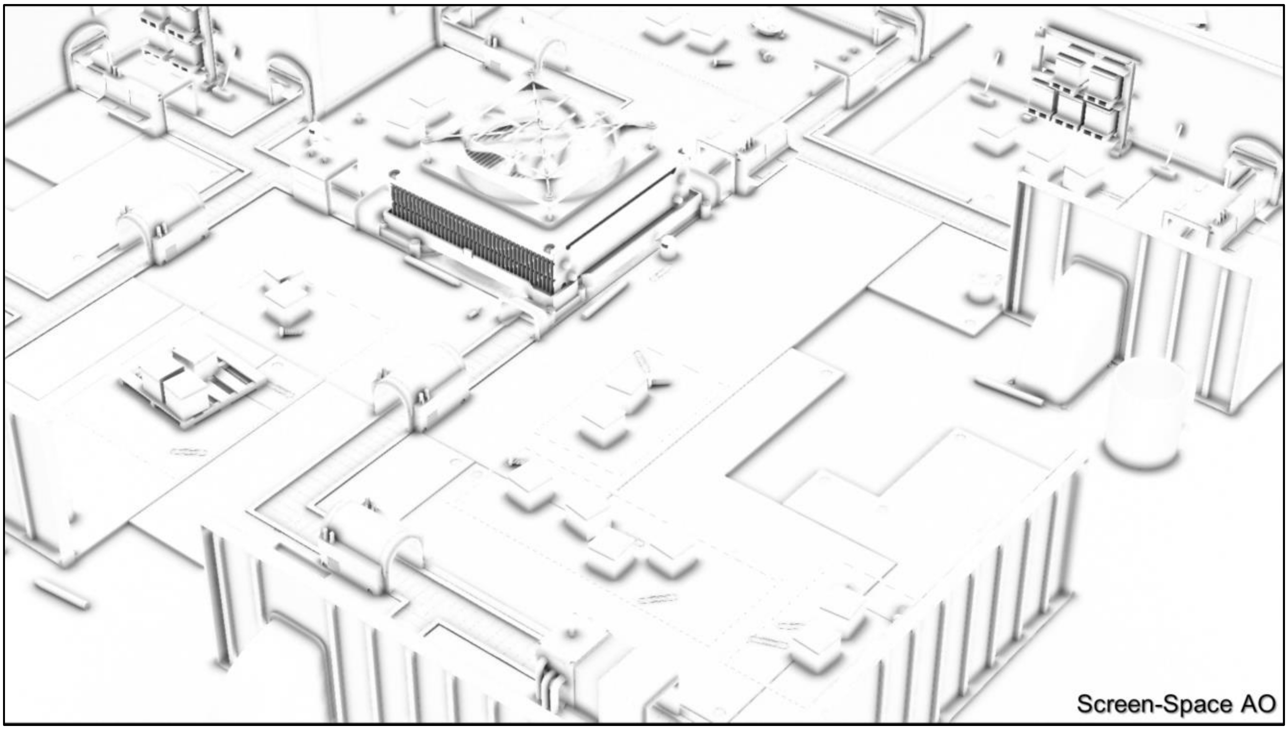
AO

Integral of the visibility function over the hemisphere Ω for the point $\bar{p}$ on a **surface** with normal $\hat{n}$ with respect to the projected solid angle

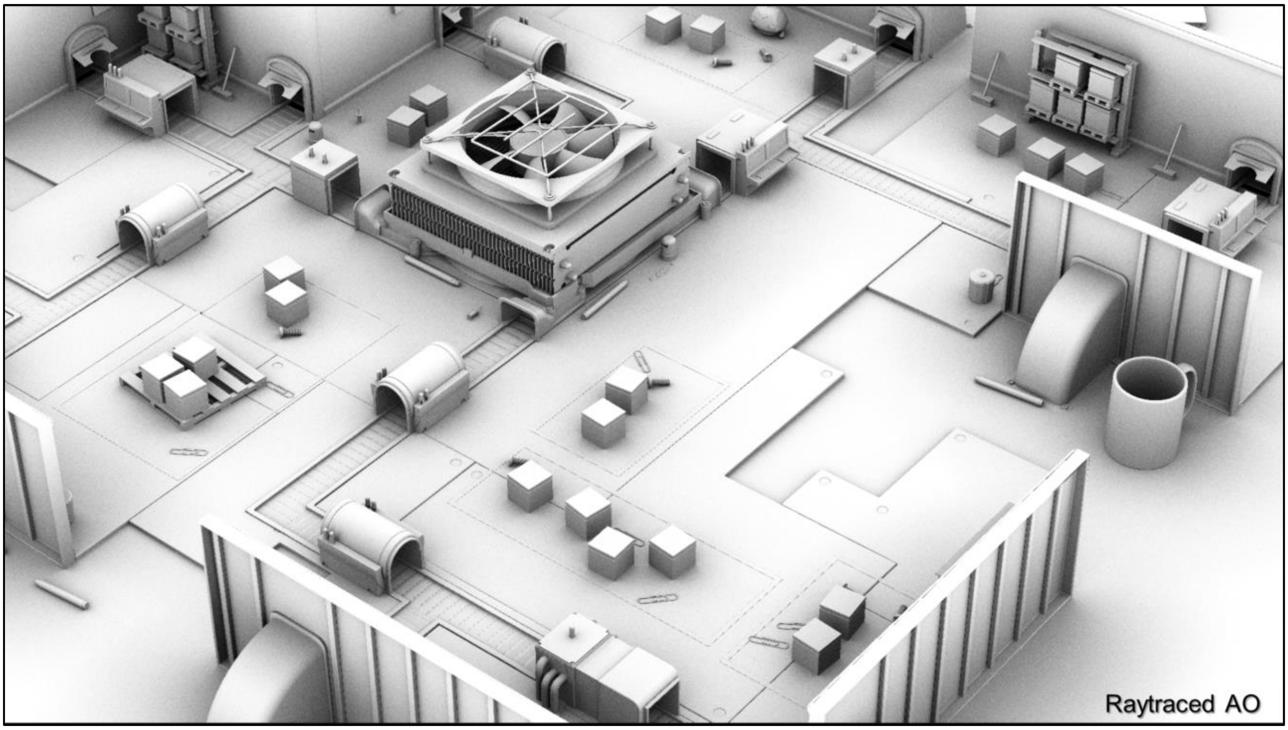
$$A_{\bar{p}} = \frac{1}{\pi}\int_{\Omega} V_{\bar{p},\hat{w}}(\hat{n}\cdot\hat{w})d\omega$$

- Random cosine hemi sampling
  - Launch from g-buffer normals
  - AO = payload.miss ? 1.0 : 0.0

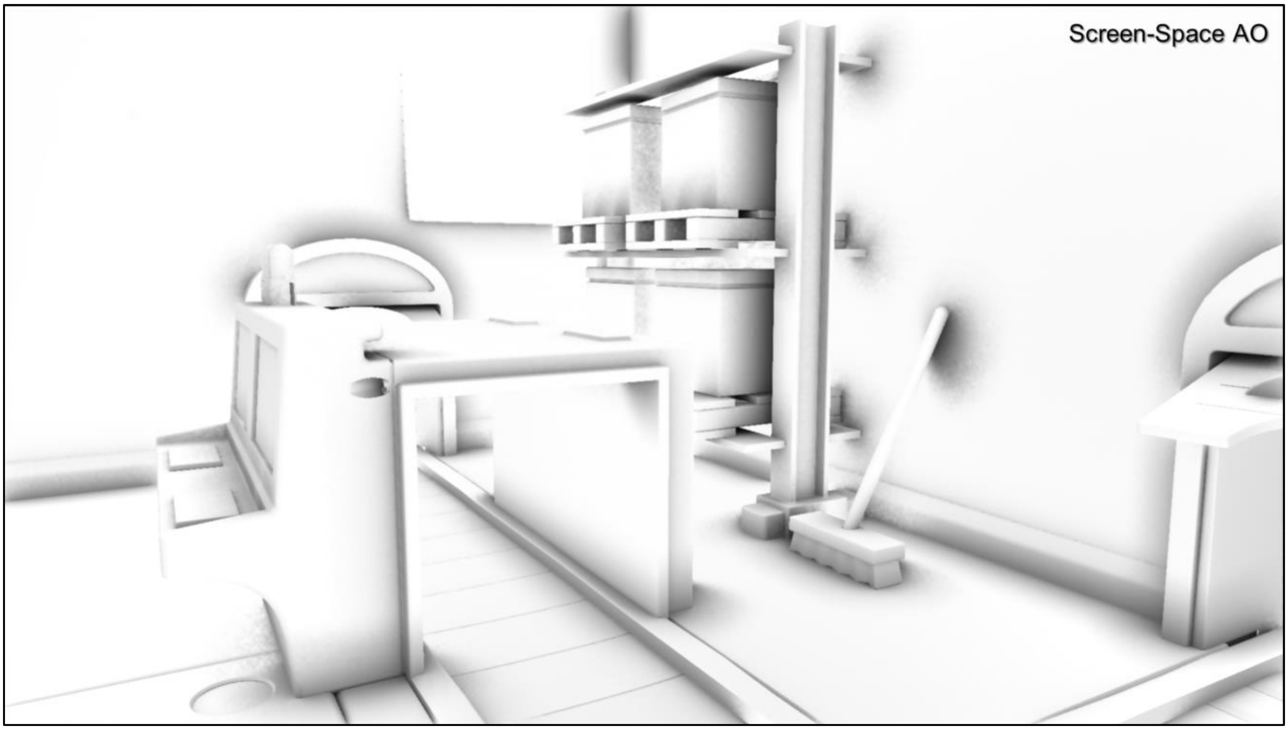
- Another technique that maps and scales well to real-time ray tracing is of course ambient occlusion, which we apply to indirect lighting.
- Being the integral of the visibility function over the hemisphere, we get more grounded results because all the random directions used during sampling actually end up in the scene, unlike with screen space techniques where rays can go outside the screen, or where the hitpoint is simply not visible.
- Just like in the literature, this is done by doing cosine hemispherical sampling around the normal.
- In our case we launch rays using the gbuffer normal, use the miss shader to figure out if we’ve hit something
- You can launch more than 1 ray per frame, but even with one ray per frame you should get some nice gradients after a few frames.
- We apply a similar filter as the one for the shadows



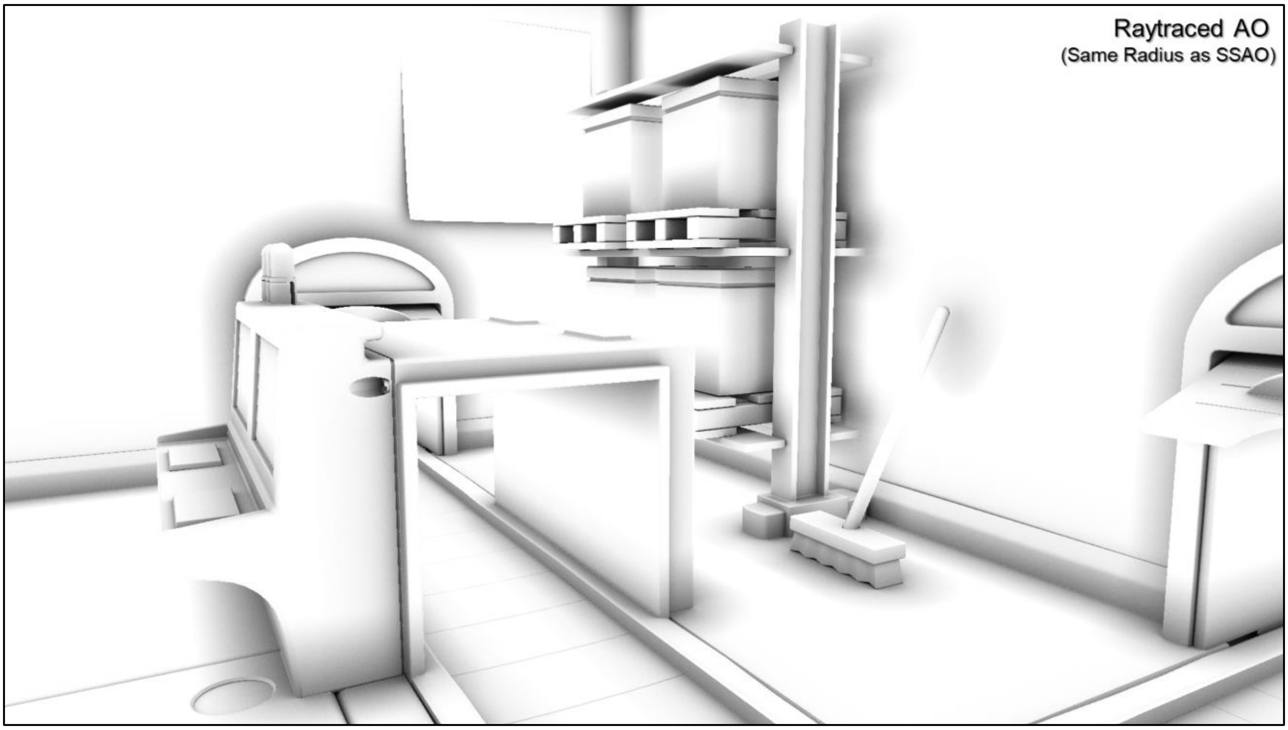
Let’s compare some results. For comparison this is screen-space AO



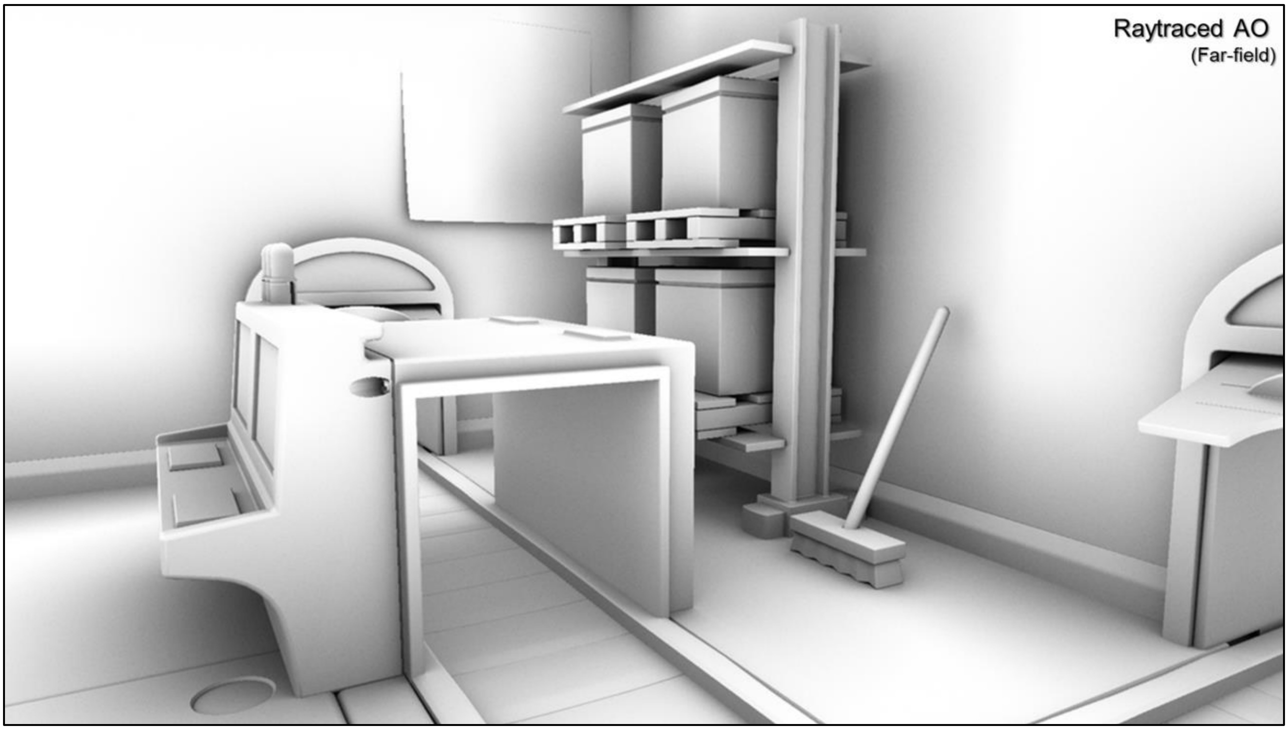
And here’s raytraced AO ☺
Looks so much more grounded!



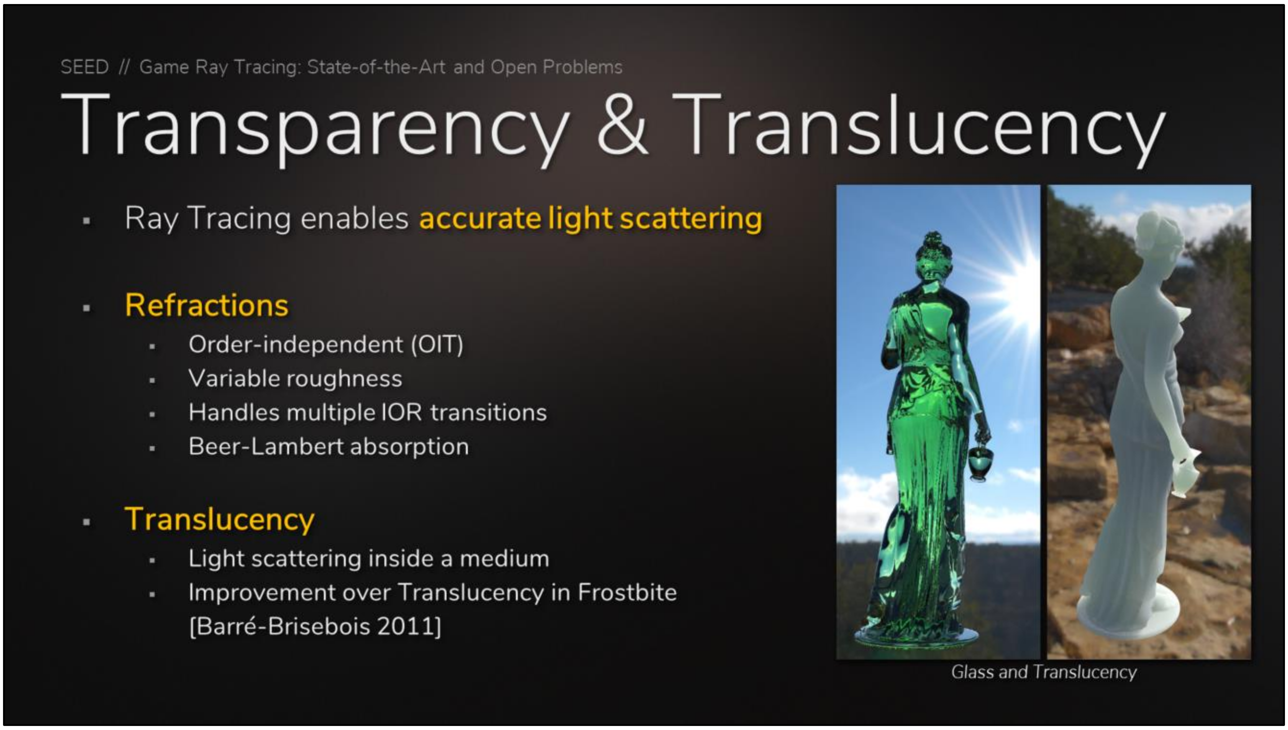
But what if we used similar ranges for the screen-space AO and the raytraced AO? Again, here’s the screenspace AO. Notice the broom against the wall with some really nice darkening



- And now raytraced AO. One can see how the missing screen space information really adds when it’s accessible through ray tracing in world space.
- TOGGLE BACK AND FORTH
- Actually, if you look carefully you’ll notice the broom is actually not touching the wall. RTAO exposed some imprecisions in our procedural placement, which SSAO did hide, so that was a funny find.



And with raytraced AO, we can even go further and have far-field AO



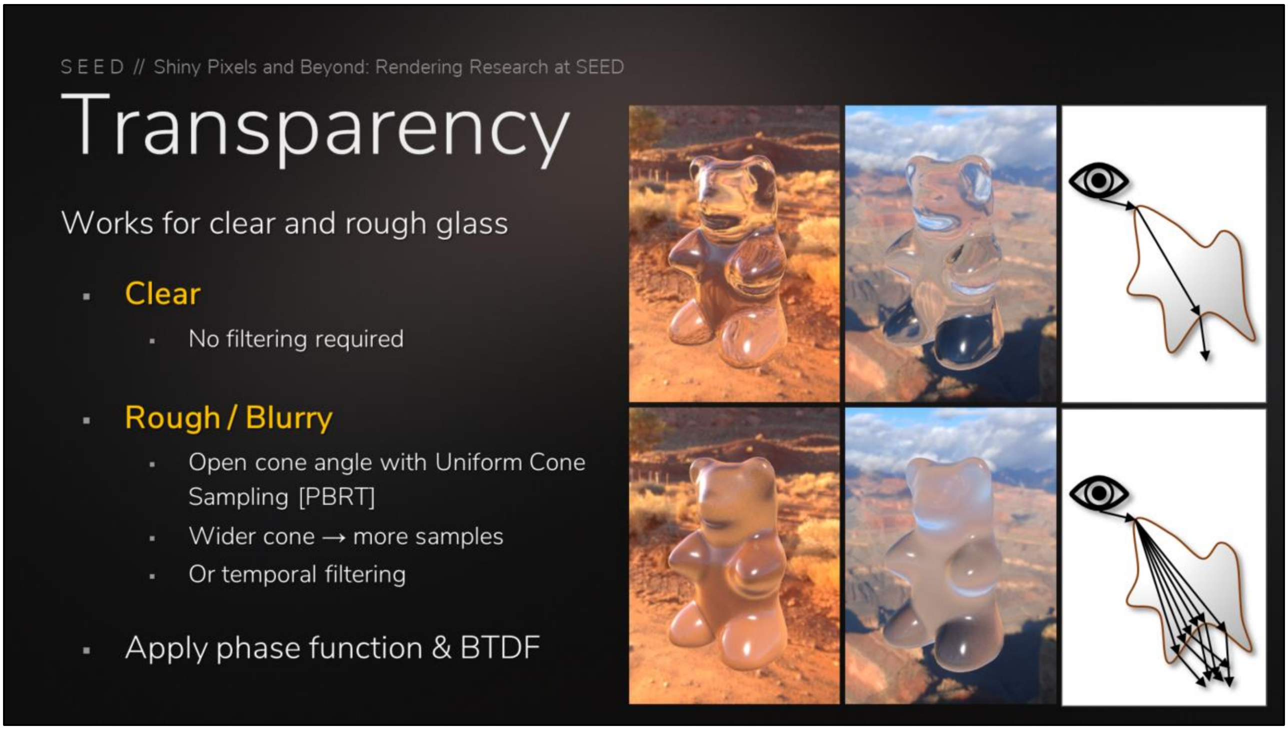
SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Transparency & Translucency

- Ray Tracing enables **accurate light scattering**
- **Refractions**
  - Order-independent (OIT)
  - Variable roughness
  - Handles multiple IOR transitions
  - Beer-Lambert absorption
- **Translucency**
  - Light scattering inside a medium
  - Improvement over Translucency in Frostbite [Barré-Brisebois 2011]

Glass and Translucency

- With ray tracing we can do proper refractions.
- We can also throw more rays, get translucency and subsurface scattering instead of the “thickness” term that I developed in EA’s engine Frostbite, which was precomputed offline and didn’t allow for self occlusion.
- We can actually have the light travel and scatter inside the medium, for all lights and taking shadows into account. This gives more convincing results, and is fully dynamic too.
- For this demo, we did it in texture space, but it can also be done in screen-space.



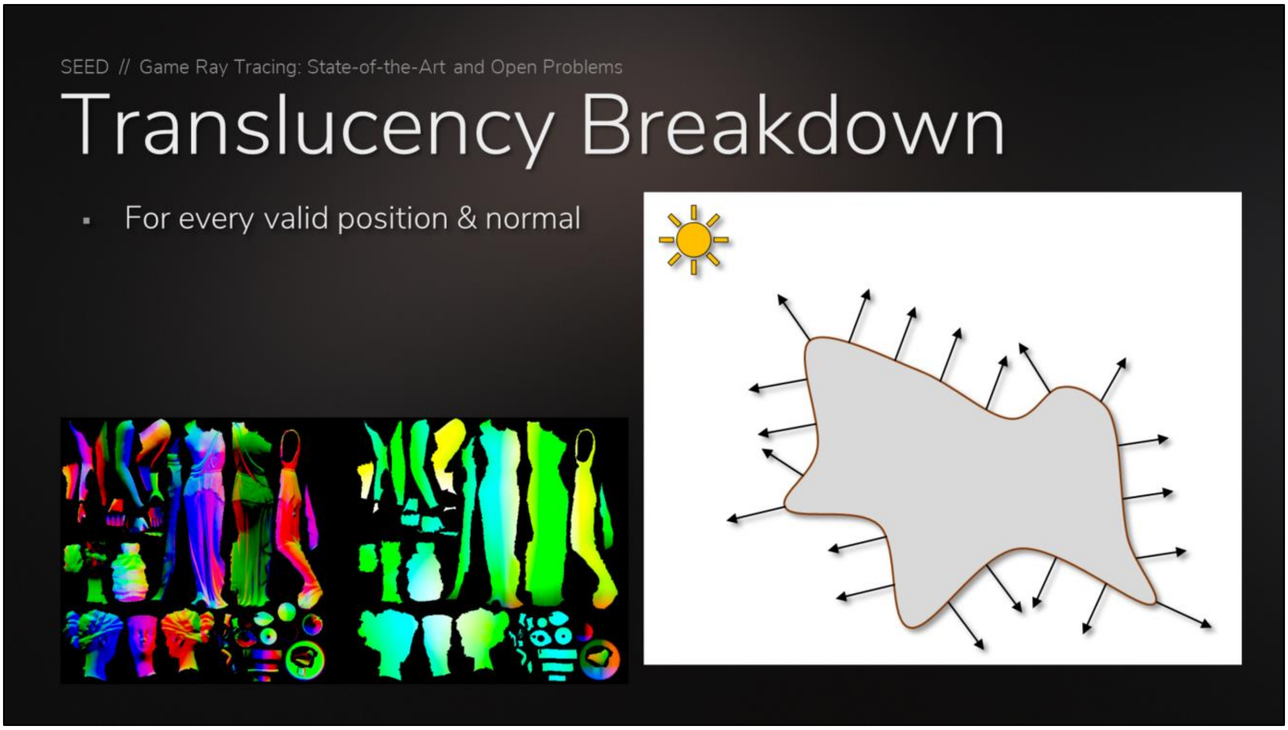
SEED // Shiny Pixels and Beyond: Rendering Research at SEED

Transparency

Works for clear and rough glass

- Clear
  - No filtering required
- Rough / Blurry
  - Open cone angle with Uniform Cone Sampling [PBRT]
  - Wider cone → more samples
  - Or temporal filtering
- Apply phase function & BTDF

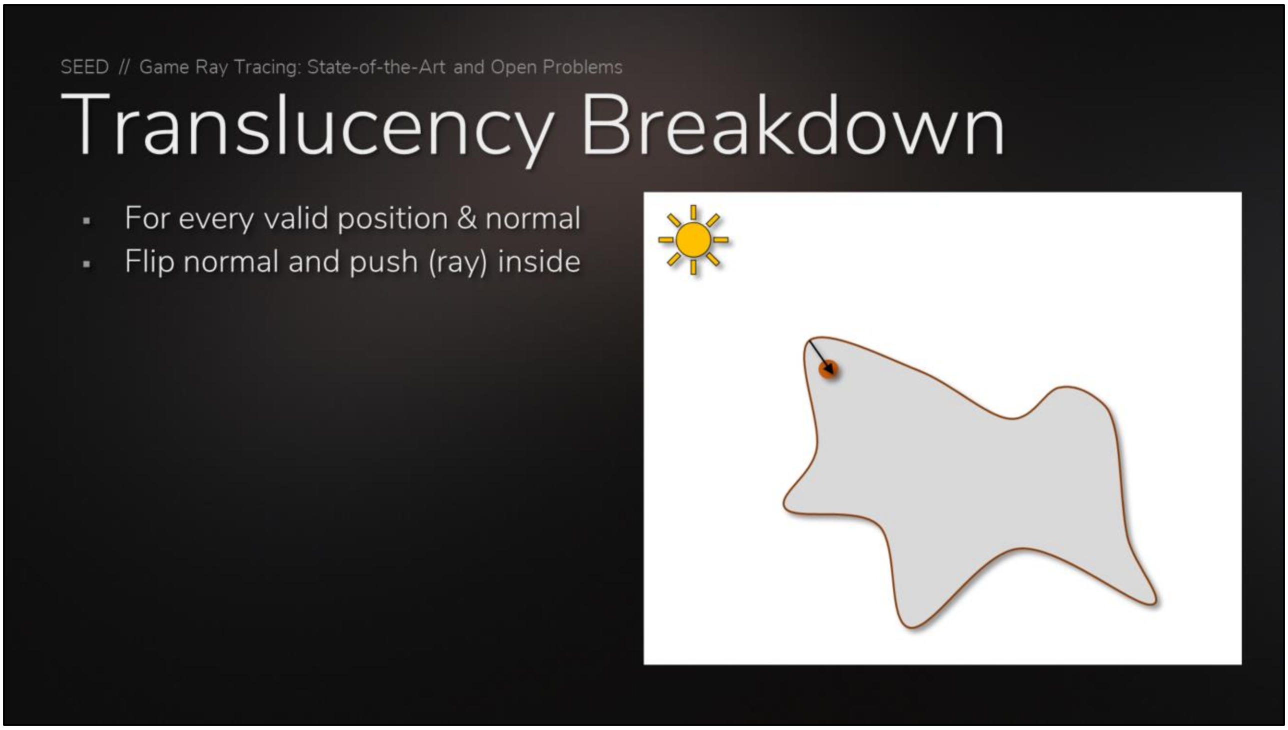
- This technique works for both clear and rough glass
- For clear glass no filtering is required
- For rough glass we open the cone angle. Obviously with rougher reflections we need more samples to get rid of noise, but one can also use temporal filtering here
- The examples here show a very simple material model for glass, but something more complex can be done if needed



Translucency Breakdown

- For every valid position & normal

Here’s a breakdown of how we compute translucency

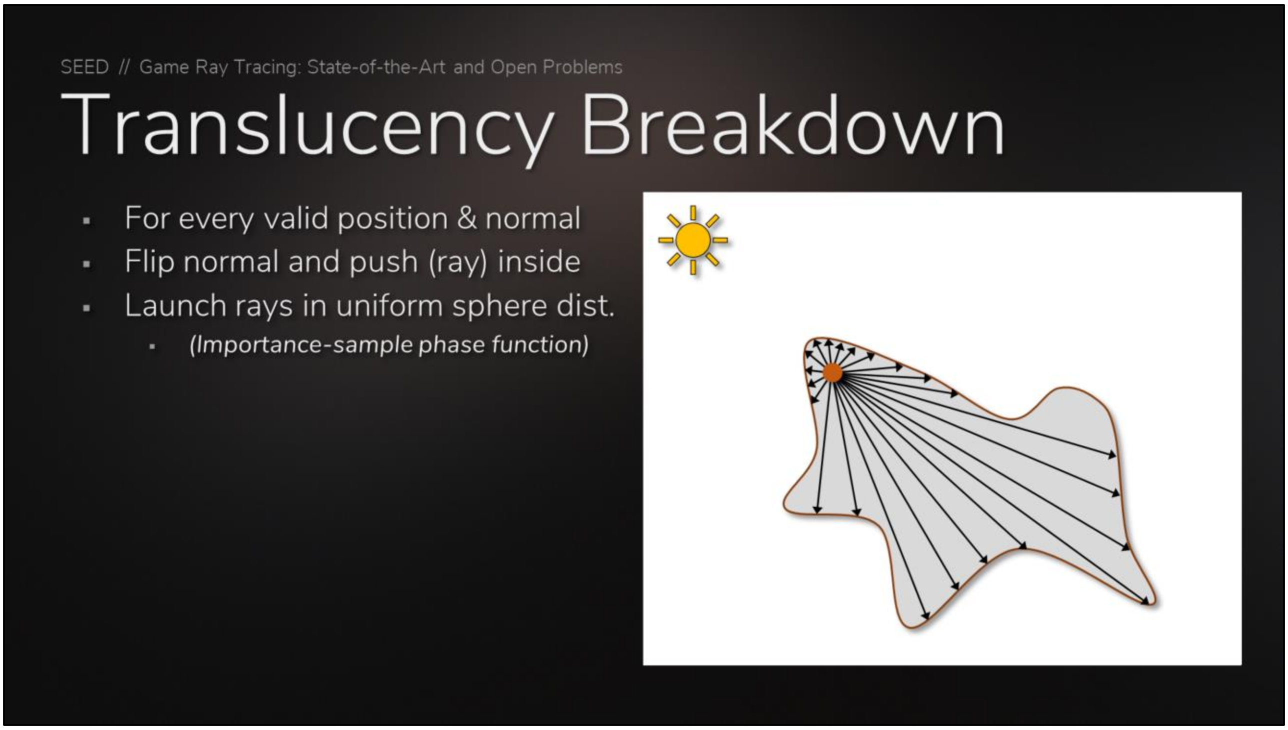


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Translucency Breakdown

- For every valid position & normal
- Flip normal and push (ray) inside

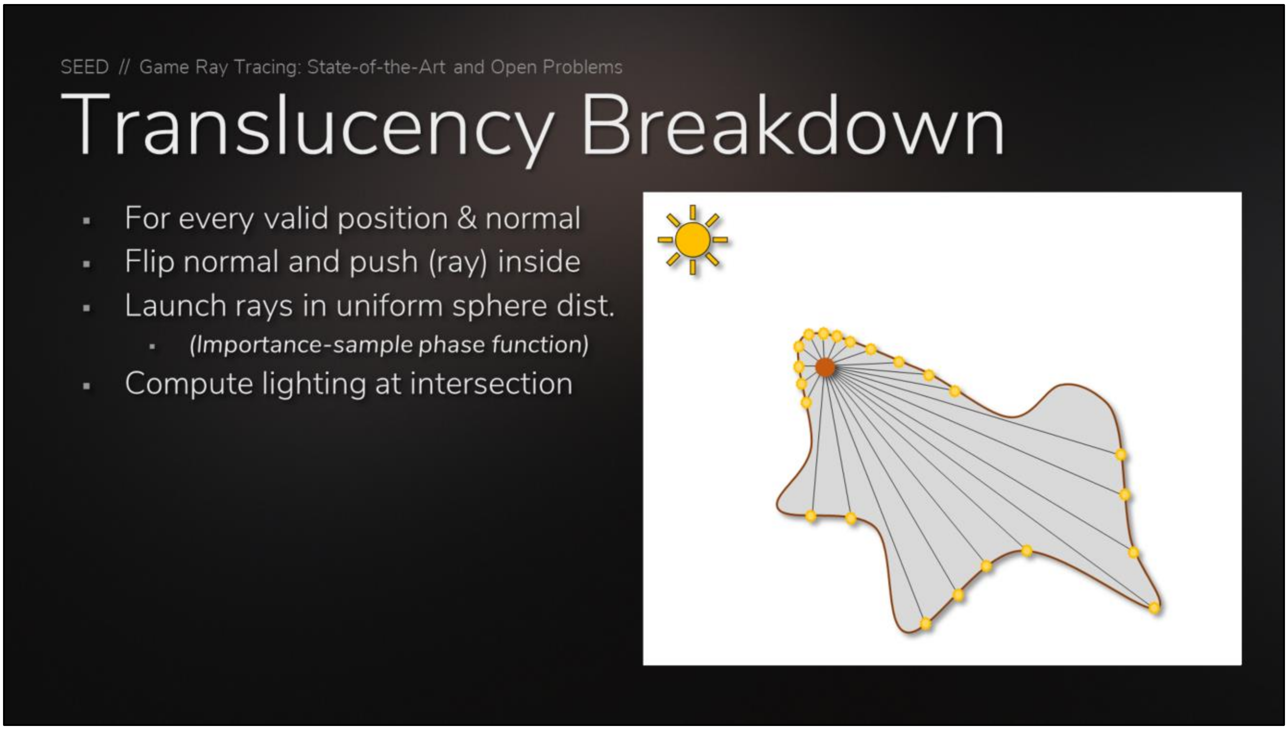
Here’s a breakdown of how we compute translucency



Translucency Breakdown

- For every valid position & normal
- Flip normal and push (ray) inside
- Launch rays in uniform sphere dist.
  - *(Importance-sample phase function)*

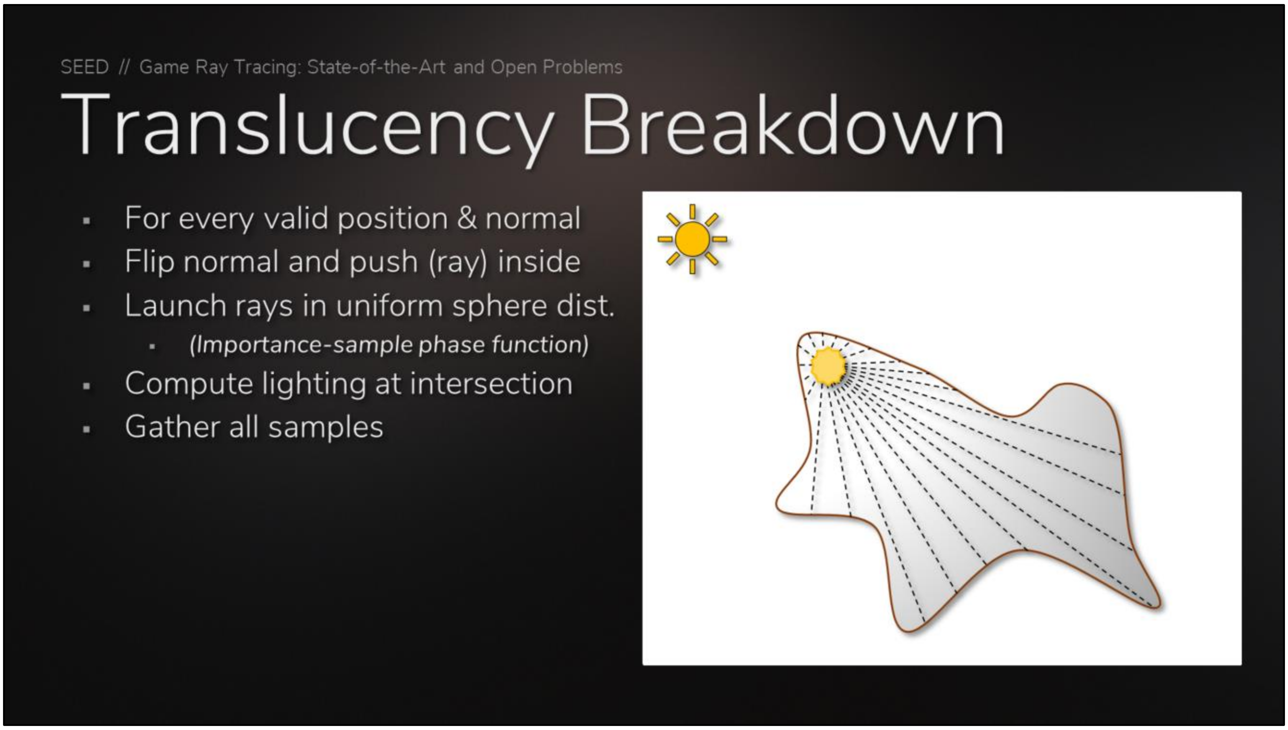
Here’s a breakdown of how we compute translucency



Translucency Breakdown

- For every valid position & normal
- Flip normal and push (ray) inside
- Launch rays in uniform sphere dist.
  - *(Importance-sample phase function)*
- Compute lighting at intersection

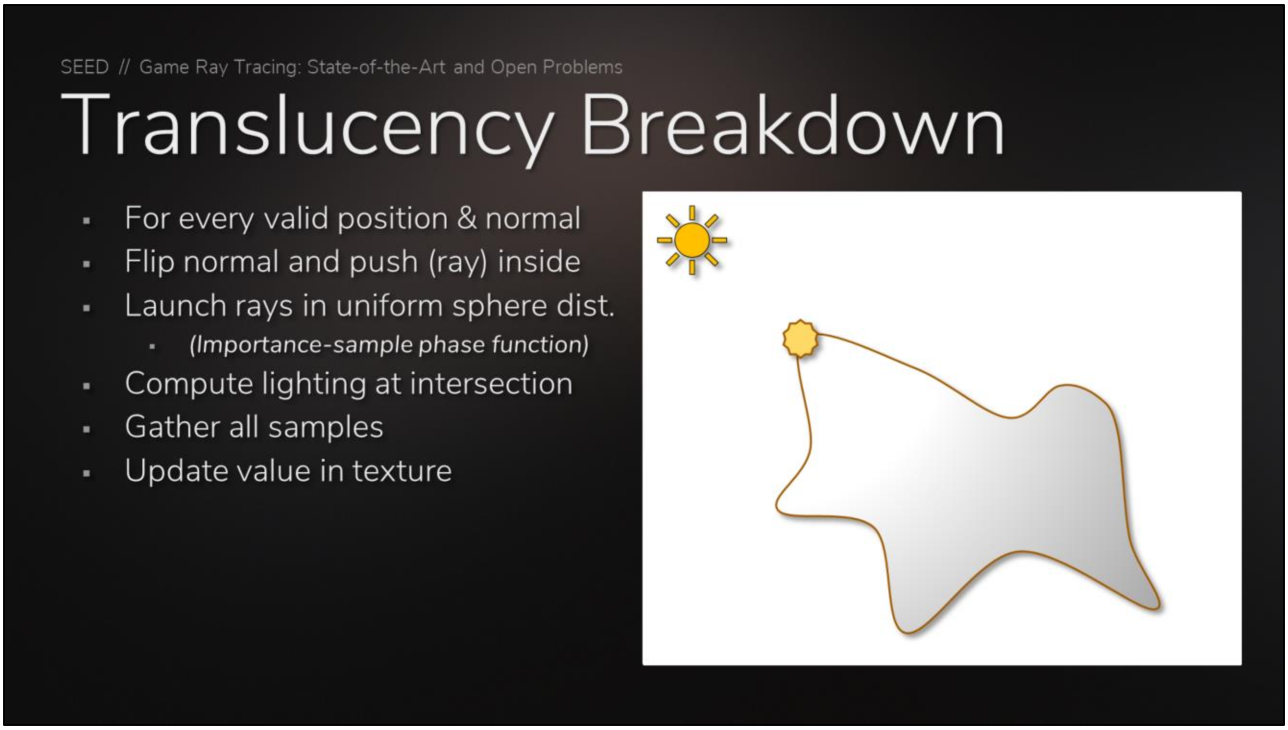
Here’s a breakdown of how we compute translucency



Translucency Breakdown

- For every valid position & normal
- Flip normal and push (ray) inside
- Launch rays in uniform sphere dist.
  - *(Importance-sample phase function)*
- Compute lighting at intersection
- Gather all samples

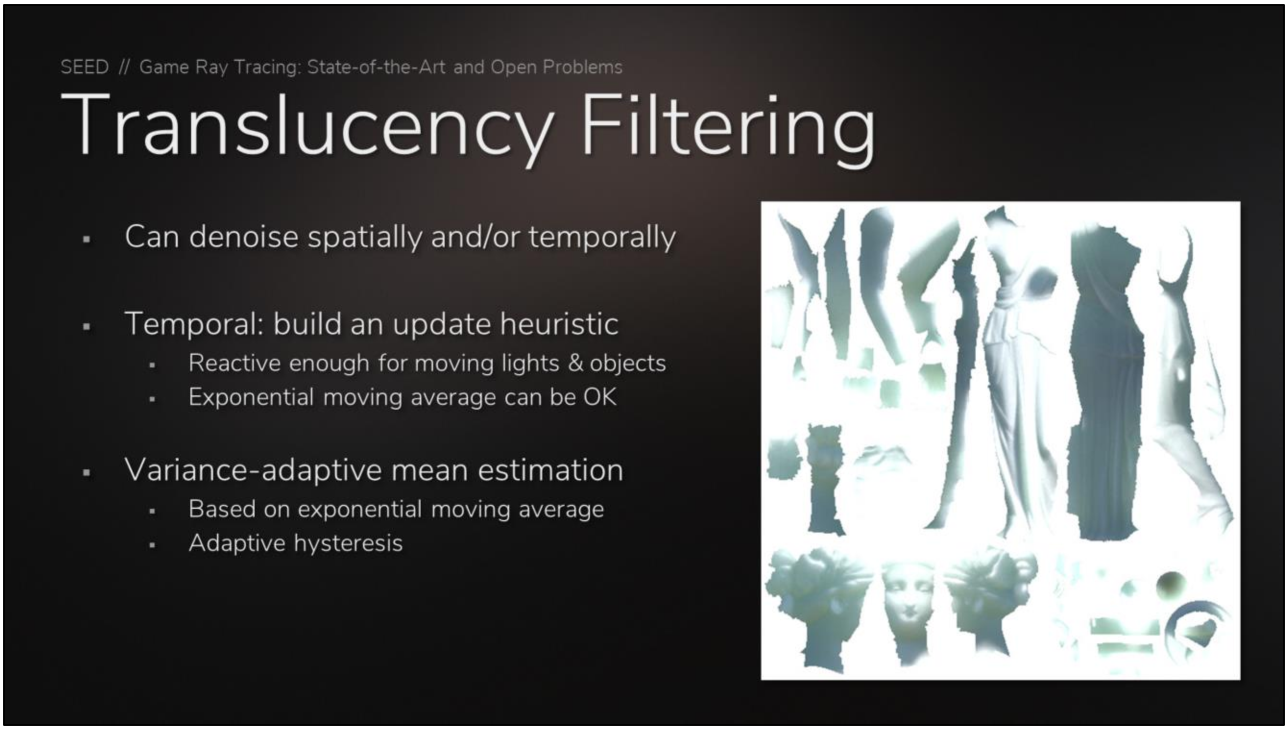
Here’s a breakdown of how we compute translucency



Translucency Breakdown

- For every valid position & normal
- Flip normal and push (ray) inside
- Launch rays in uniform sphere dist.
  - *(Importance-sample phase function)*
- Compute lighting at intersection
- Gather all samples
- Update value in texture

Here’s a breakdown of how we compute translucency

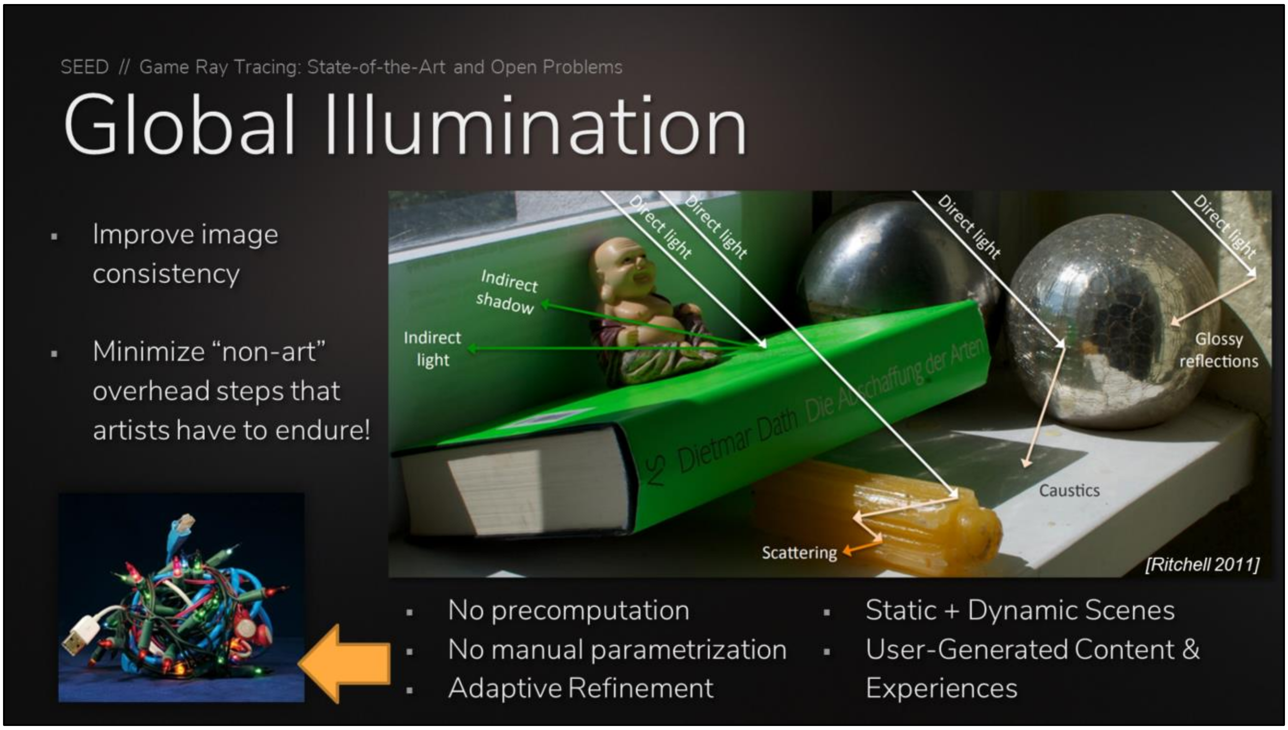


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Translucency Filtering

- Can denoise spatially and/or temporally
- Temporal: build an update heuristic
  - Reactive enough for moving lights & objects
  - Exponential moving average can be OK
- Variance-adaptive mean estimation
  - Based on exponential moving average
  - Adaptive hysteresis

- We let the results converge over multiple frames via temporal accumulation. Spatial filtering can be used as well, although we didn’t hit enough noise to make it worthwhile.
- Since lighting conditions can change with objects moving, a temporal filter needs to be able to adapt to them. Depending on your use case, a simple exponential moving average can do the trick.
- We use a slightly fancier, adaptive temporal filter based on exponential averaging. It varies its hysteresis to quickly re-converge to dynamically changing conditions.



SEED // Game Ray Tracing: State-of-the-Art and Open Problems

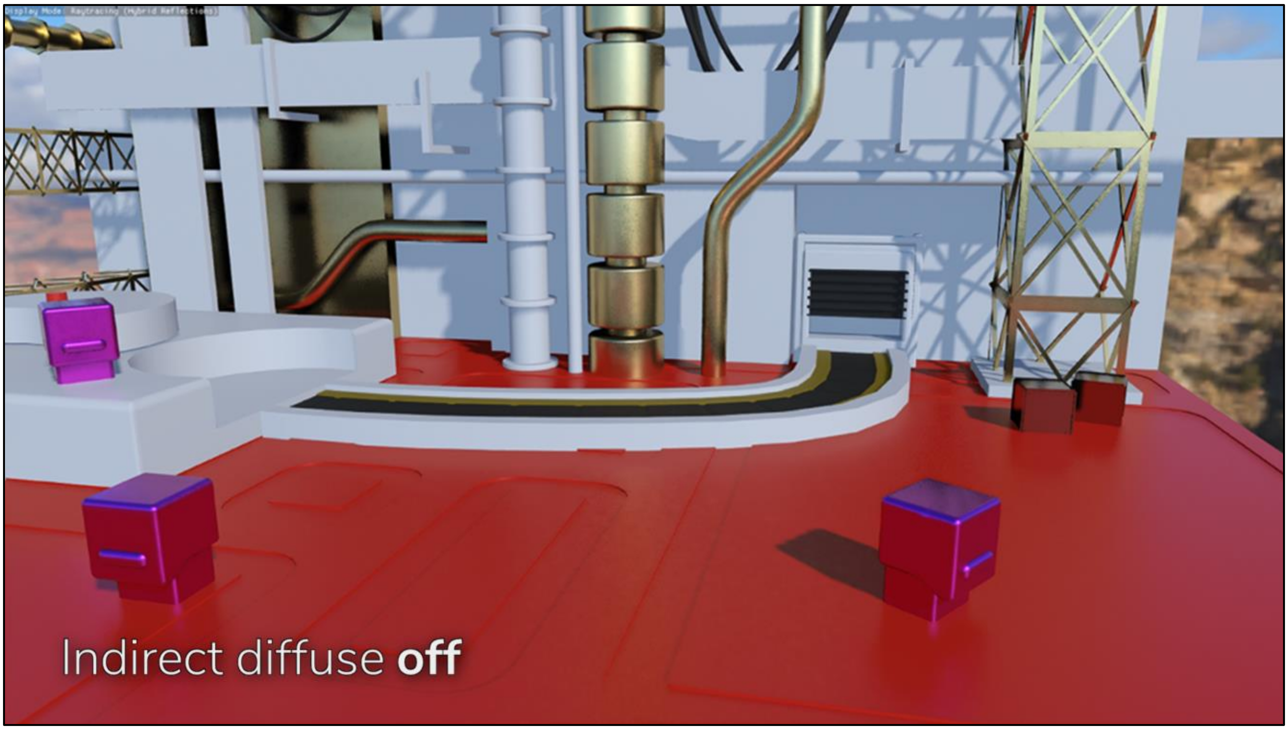
Global Illumination

- Improve image consistency
- Minimize “non-art” overhead steps that artists have to endure!

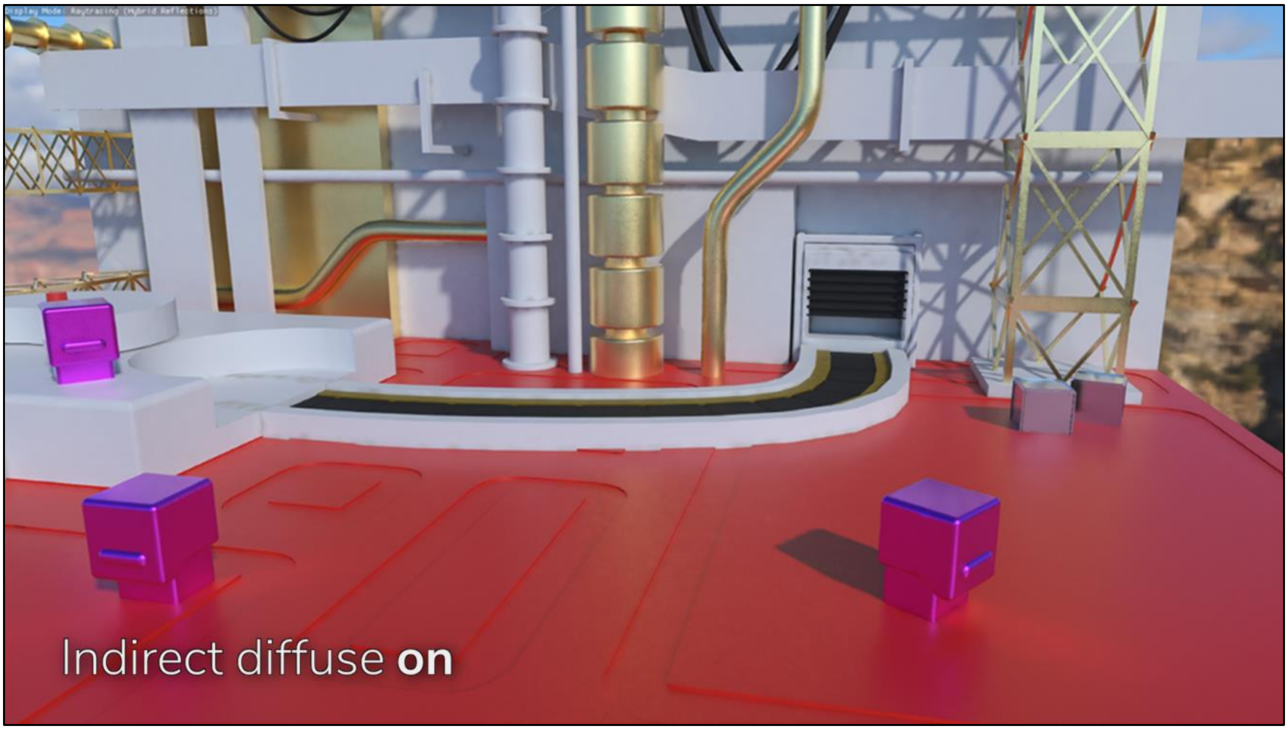
[Ritchell 2011]

- No precomputation
- No manual parametrization
- Adaptive Refinement
- Static + Dynamic Scenes
- User-Generated Content & Experiences

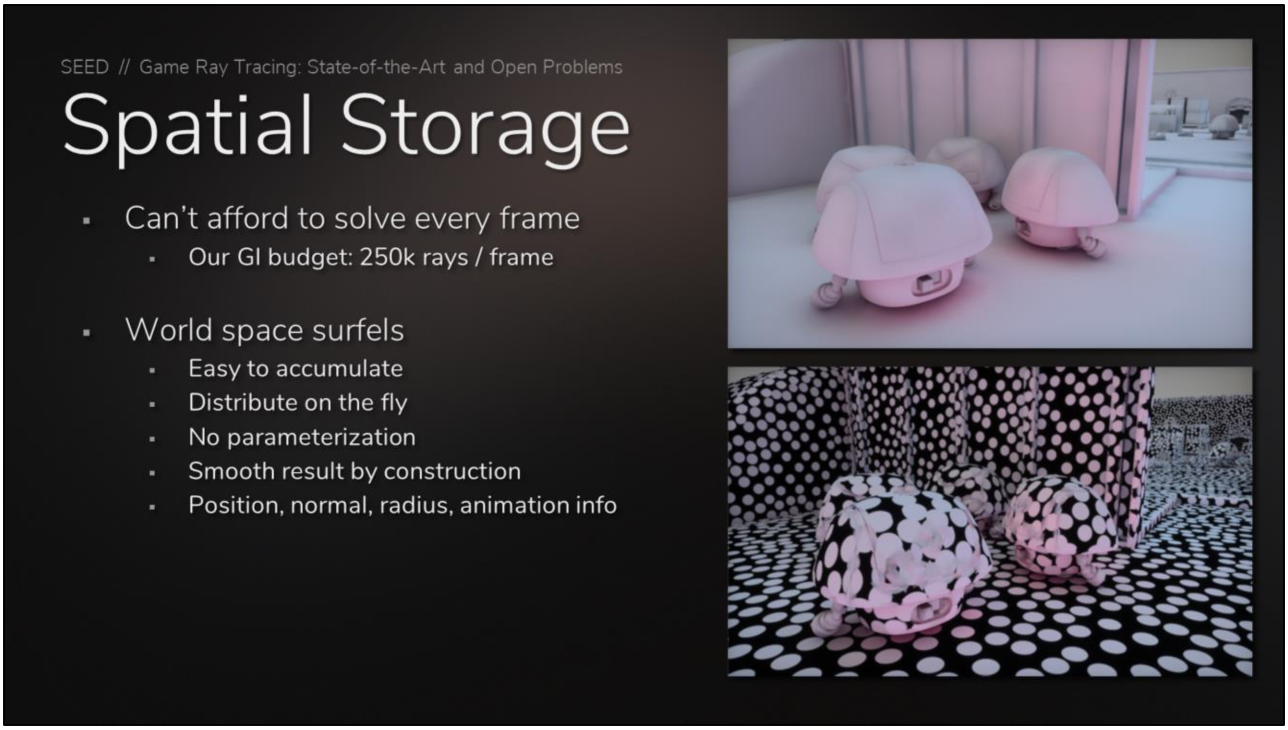
- We’ve covered indirect specular and transmission, so let’s talk about the rest of light transport. In this case I mean indirect lighting applied in a diffuse manner to scene surfaces.
- Great materials require great light transport. Of course ray tracing makes it possible to implement it better than ever.
- GI makes scene elements fit with each other, and makes the life of artists better, as it reduces the need to manually place artificial lights as many games have shipped with, or even touch up surface colors and destroy PBR.
- Or even worst: negative lights. What the hell is that anyway!
- We set out to create a solution that was very dynamic, without the need for any precomputation or even UVs. It had to support dynamic and static scenes, and we wanted it to refine to a high quality result, at least in static regions.
- And in the future we’ll need to support static + dynamic scenes, user generated content, arbitrary content that you can’t ask artists to clean-up and preprocess.



- An initial PICA PICA test scene with indirect diffuse off



- And now on. Notice the bounce in the back and in the front on the robots

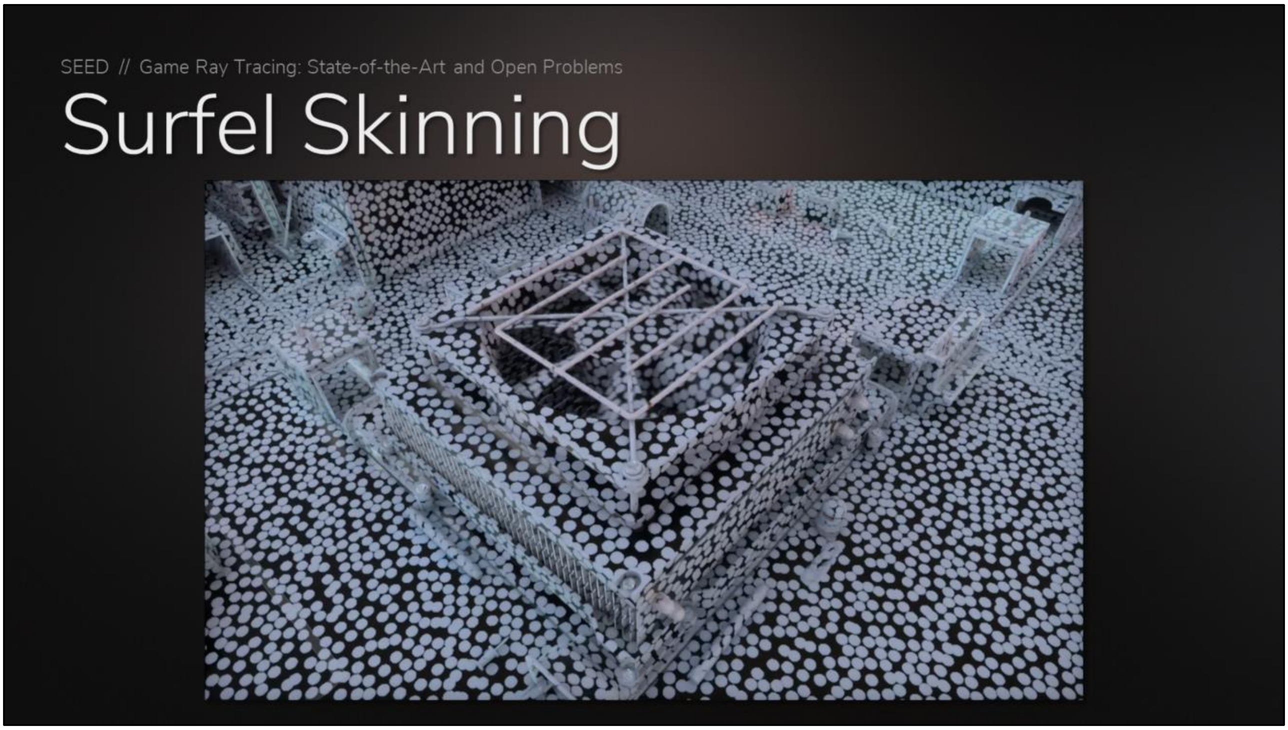


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Spatial Storage

- Can't afford to solve every frame
  - Our GI budget: 250k rays / frame
- World space surfels
  - Easy to accumulate
  - Distribute on the fly
  - No parameterization
  - Smooth result by construction
  - Position, normal, radius, animation info

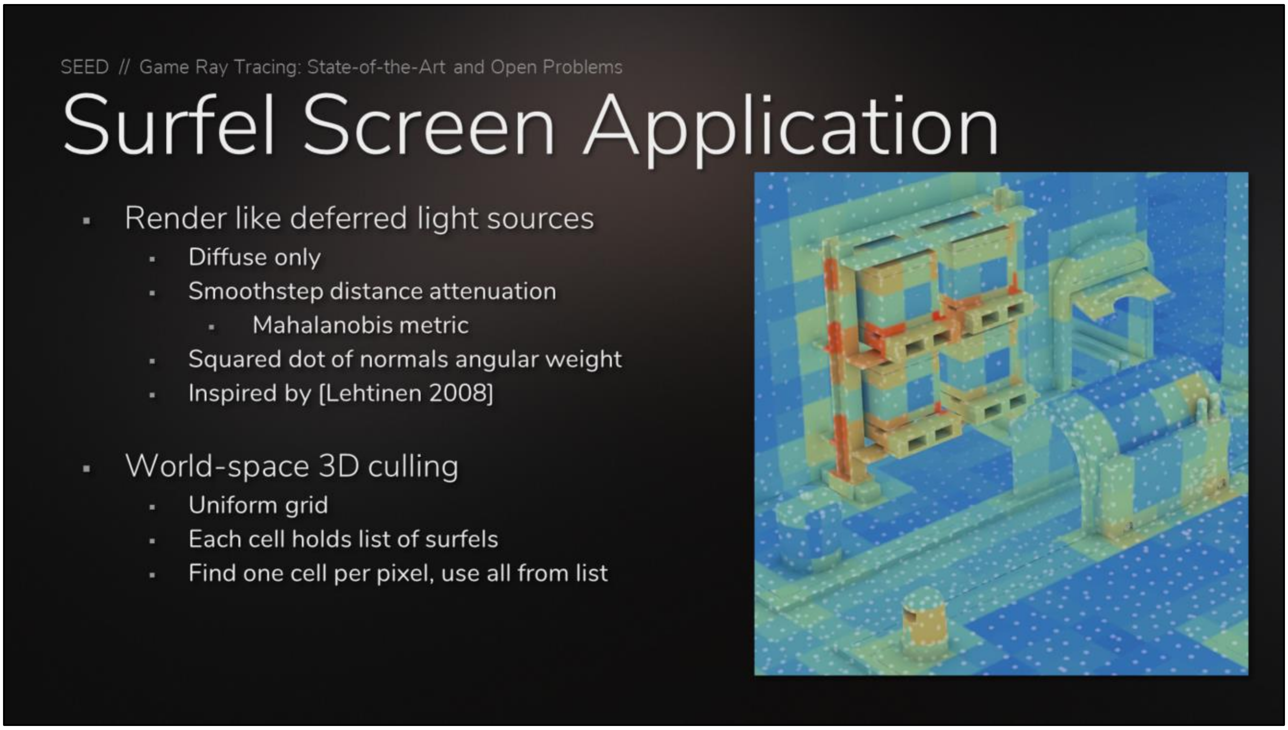
- When it comes to indirect lighting, or global illumination we can't afford to re-solve every frame
- We set ourselves a 250K rays per frame budget
- For PICA PICA we wanted a technique that doesn't require precomputation, parametrization (such as Uvs and mesh proxies), works for both dynamic and static scenes
- Basically, no step to piss-off artists and to be honest it's 2018, I think it's time. And in the future we want to support arbitrary content from anyone, so can't expect them to unwrap meshes or build mesh proxies.
- And so we ended up using world space surfels that we spawn on what you see



SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Surfel Skinning

- To support animated objects, surfels remember the object on which they were spawned, and are updated every frame to move with it.
- This does pose a challenge to temporal accumulation, but it’s not as severe as ghosting from screen-space techniques.

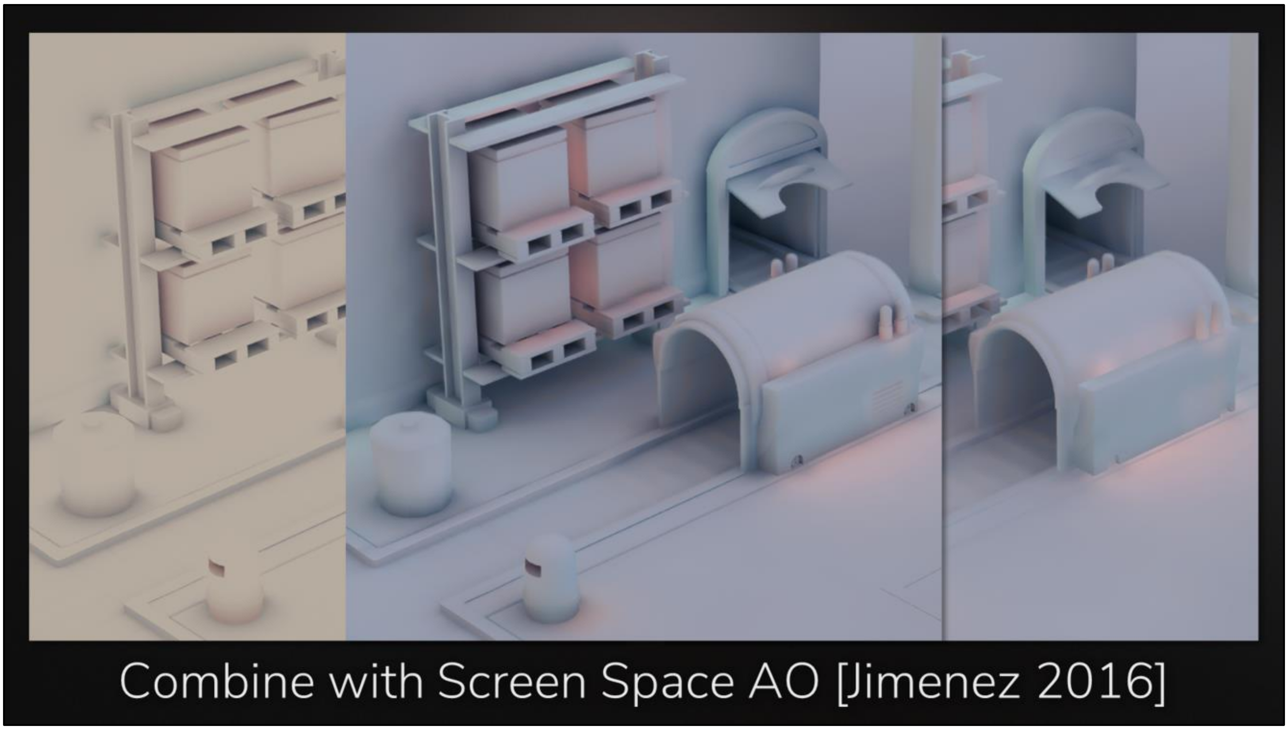


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Surfel Screen Application

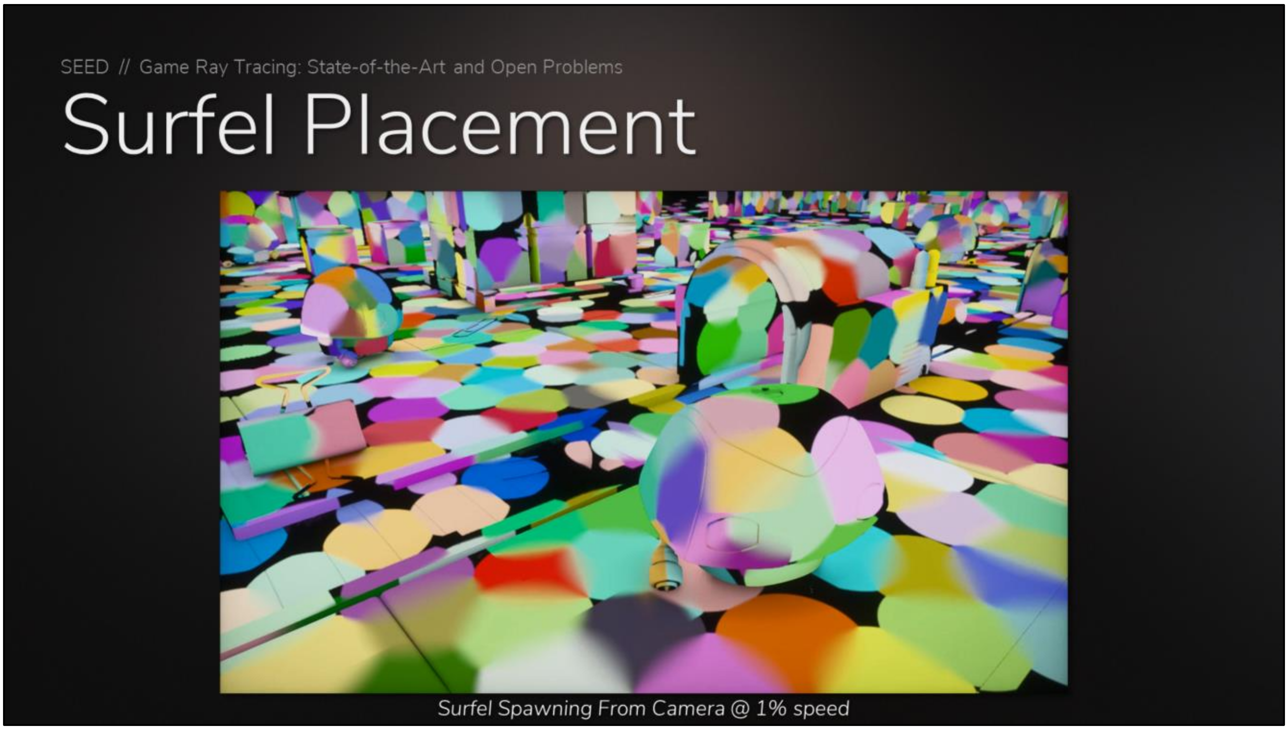
- Render like deferred light sources
  - Diffuse only
  - Smoothstep distance attenuation
    - Mahalanobis metric
  - Squared dot of normals angular weight
  - Inspired by [Lehtinen 2008]
- World-space 3D culling
  - Uniform grid
  - Each cell holds list of surfels
  - Find one cell per pixel, use all from list

- To apply surfels to the screen, we render them in a similar way to light sources.
- We use smoothstep for distance attenuation, with the Mahalanobis metric to squash them in the normal direction.
- We also have angular falloff, but each surfel’s payload is just irradiance, without any directionality.
- Also similarly to deferred lights, we have a surfel culling system.
- In order to be able to query the GI solution anywhere, we use a world-space data structure.
- For simplicity, it’s a grid in which every cell stores a list of surfels. Each pixel or point in space can then look up the containing cell, and find all relevant surfels.



Combine with Screen Space AO [Jimenez 2016]

- The GI has limited resolution, and lacks high frequency detail.
- We augment it with screen-space ambient occlusion implemented after “Ground Truth AO” from Jorge Jimenez.
- We use the colored multi-bounce variant, as it helps retain the warmth in our toy-like scenes.

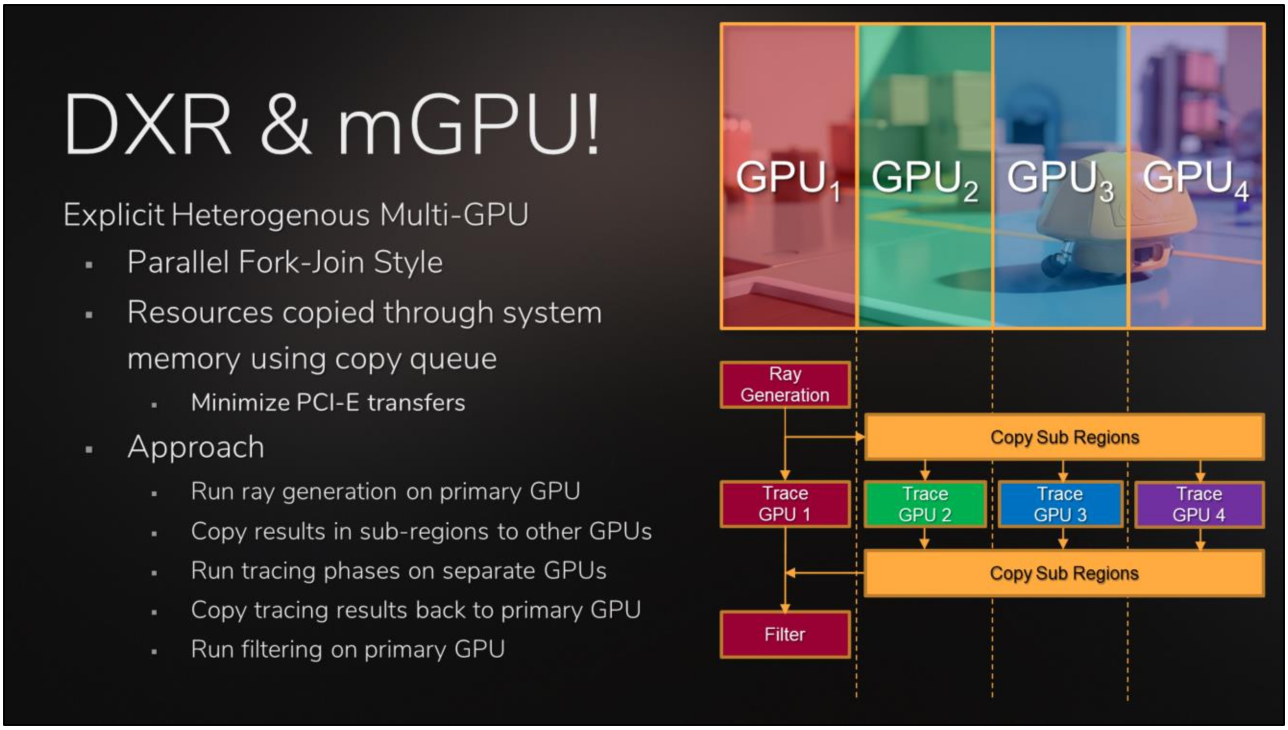


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Surfel Placement

Surfel Spawning From Camera @ 1% speed

- As I mentioned, surfels are spawned on the fly, and require no precomputation.
- Here’s a clip of the process in action at around one percent speed.
- As you can see, the distribution is quite even and resembles Poisson disk.
- Haven’t covered irradiance transfer here, but for the sake of time you should check tom’s slides from Digital Dragons 2018.



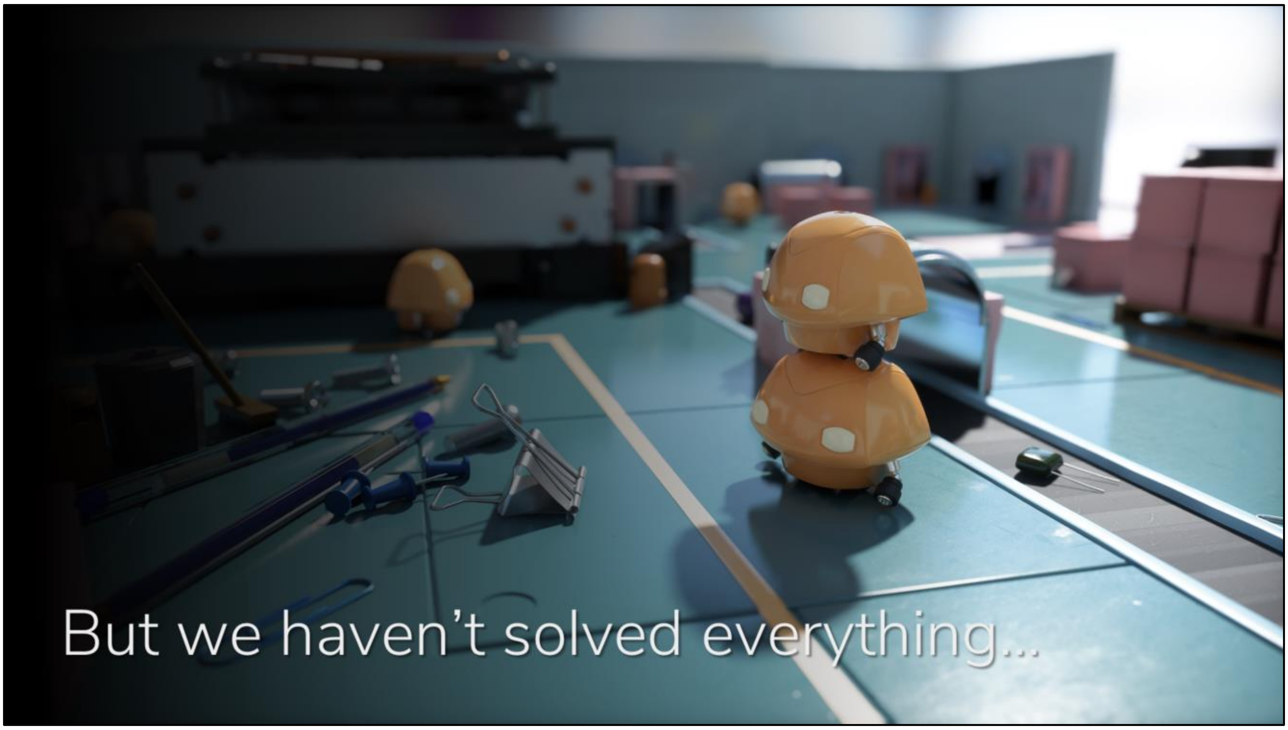
DXR & mGPU!

Explicit Heterogenous Multi-GPU

- Parallel Fork-Join Style
- Resources copied through system memory using copy queue
  - Minimize PCI-E transfers
- Approach
  - Run ray generation on primary GPU
  - Copy results in sub-regions to other GPUs
  - Run tracing phases on separate GPUs
  - Copy tracing results back to primary GPU
  - Run filtering on primary GPU

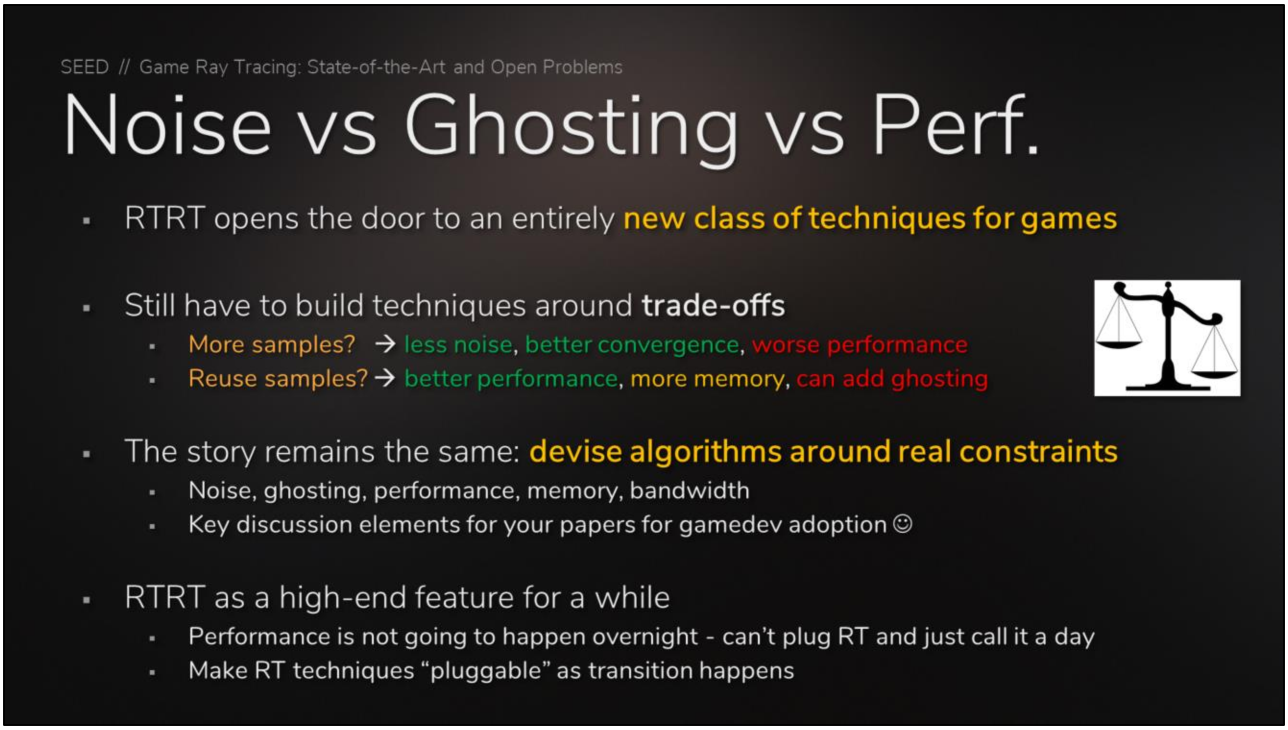
A bit of a side note, but for n our demo we also explored doing ray tracing on multiple GPUs. The primary GPU acts as the arbiter, splitting the work for secondary devices and finishes the final work.

- Run ray generation phase on primary GPU for the entire screen
- Copy ray generation texture in sub-regions to other GPUs
- Run tracing phases on other GPUs (just a sub-region of the entire screen on each GPU)
- Copy tracing results back in sub-regions from other GPUs
- Filtering is then done only on primary GPU. This allows us to mitigate temporal problems.



But we haven’t solved everything...

- We obviously haven’t solved everything, and still have a bunch of topics we need to tackle as an industry that I would like to discuss with you.

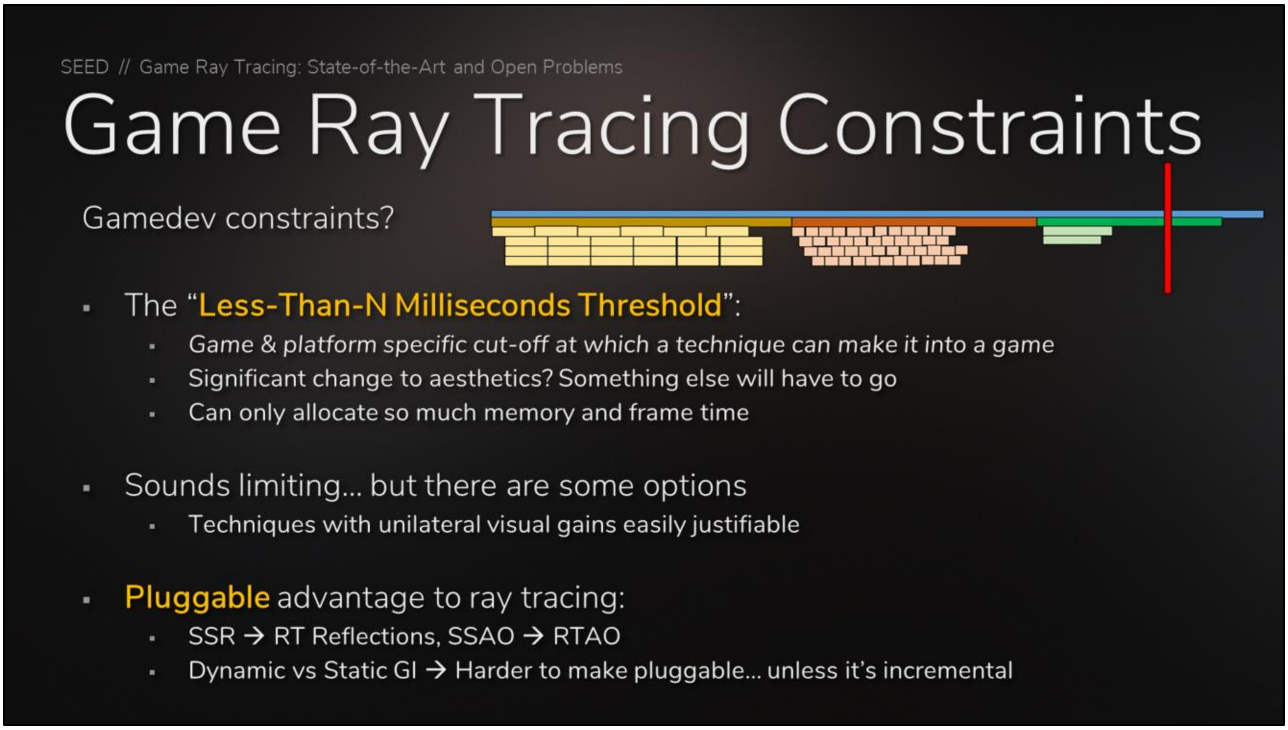


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Noise vs Ghosting vs Perf.

- RTRT opens the door to an entirely **new class of techniques for games**
- Still have to build techniques around **trade-offs**
  - More samples? → less noise, better convergence, worse performance
  - Reuse samples? → better performance, more memory, can add ghosting
- The story remains the same: **devise algorithms around real constraints**
  - Noise, ghosting, performance, memory, bandwidth
  - Key discussion elements for your papers for gamedev adoption ☺
- RTRT as a high-end feature for a while
  - Performance is not going to happen overnight - can't plug RT and just call it a day
  - Make RT techniques "pluggable" as transition happens

- Real-time ray tracing opens the door to an entirely new class of techniques for games, which is awesome!
- Not automagical: we still have to build around trade-offs. For example…
- Still need to devise techniques around real constraints, such as: …
- For researchers, this becomes a key discussion elements for your papers, for gamedev adoption ☺
- And so for now it feels like RTRT will be seen as a high-end feature, for some amount of time at least
- Obviously performance won't happen over night
- As the transition happens, need to make those techniques swappable or pluggable, depending on the hardware customers have, so that other SKUs don't suffer: SSR → RT Reflections, SSAO → RTAO



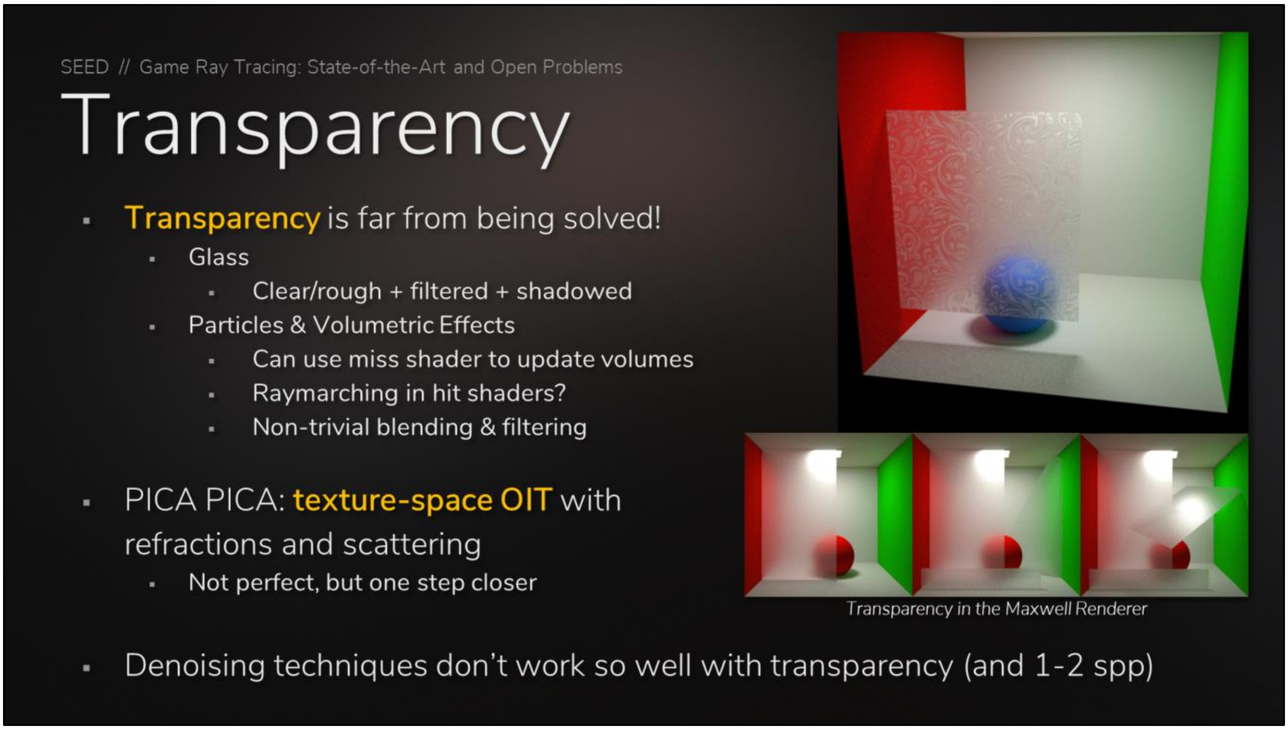
SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Game Ray Tracing Constraints

Gamedev constraints?

- The “**Less-Than-N Milliseconds Threshold**”:
  - *Game & platform specific cut-off at which a technique can make it into a game*
  - Significant change to aesthetics? Something else will have to go
  - Can only allocate so much memory and frame time
- Sounds limiting… but there are some options
  - Techniques with unilateral visual gains easily justifiable
- **Pluggable** advantage to ray tracing:
  - SSR → RT Reflections, SSAO → RTAO
  - Dynamic vs Static GI → Harder to make pluggable… unless it’s incremental

- So what do I mean by gamedev constraints?
- In gamedev we have this thing called the “less than N milliseconds threshold”, which is basically a game and platform specific cutoff at which a technique passes all the gates (technical, art, production) and can make it into a game.
- Assume 30-60FPS, so 16ms to 33ms per frame which is shared between all. A lot of techniques run between 1-3ms. Sometimes less, sometimes a bit more. It all depends on the kind of game you’re shipping, but generally these are the kind of ranges we look at
- Limitations on memory as well as well as production constraints: for example, will this help or make the art process more tedious?
- So, if your technique is awesome, passes the thresholds, and brings significant value, awesome!
- But if the frame is full, it’s a different discussion.
- This sounds quite limiting, but there are some options around this. Especially on PC since it’s easier to tailor for high-end features, unlike on consoles which are more fixed. Though a bit of a different story now with PS4 pro and Xbox1X
- As mentioned there’s a pluggable advantage to ray tracing: one can swap SSR with RT reflections, and SSAO with Raytraced AO, so that allows us to have two profiles, a normal one and a high-end one
- Now for some other techniques it might be a bit more difficult. For example if your game depends on raytraced real-time GI, where you have gameplay around dynamic lights turning on or off, it might be harder to make it pluggable, unless it’s incremental.



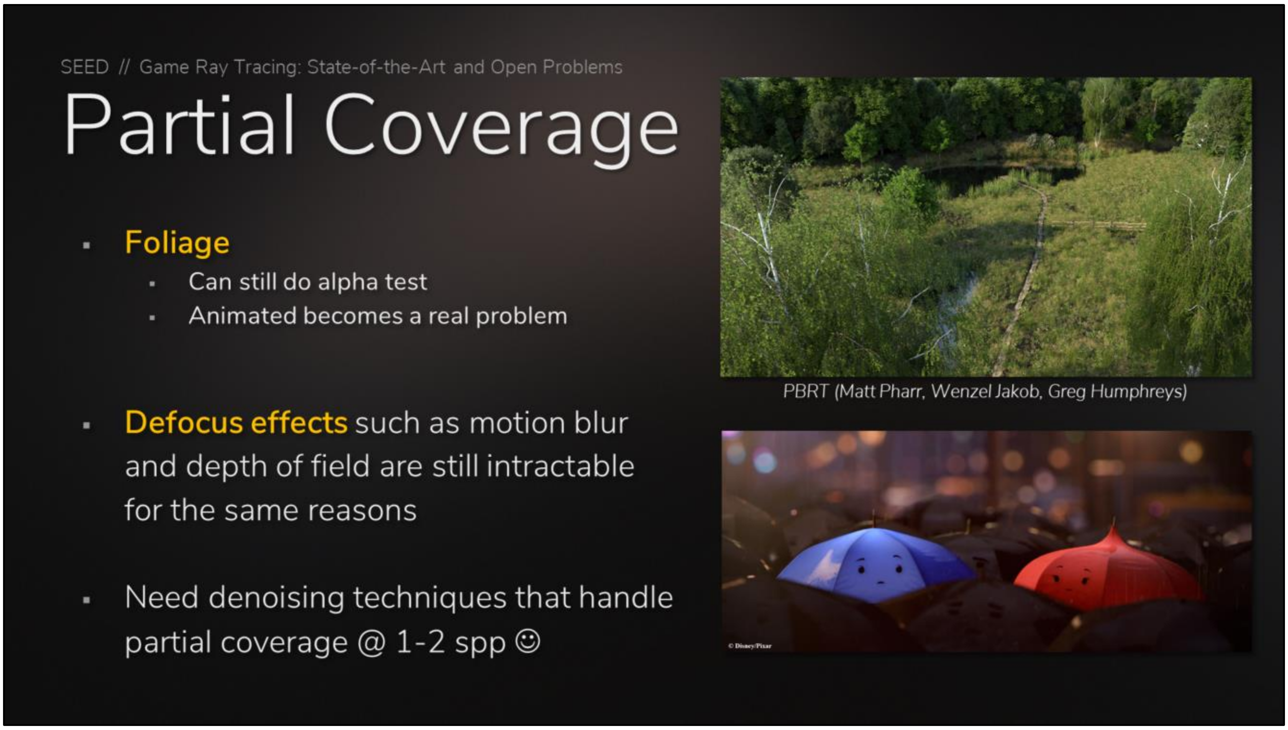
SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Transparency

- **Transparency** is far from being solved!
  - Glass
    - Clear/rough + filtered + shadowed
  - Particles & Volumetric Effects
    - Can use miss shader to update volumes
    - Raymarching in hit shaders?
    - Non-trivial blending & filtering
- PICA PICA: **texture-space OIT** with refractions and scattering
  - Not perfect, but one step closer
- Denoising techniques don't work so well with transparency (and 1-2 spp)

Transparency in the Maxwell Renderer

- Here's another open problem: transparency and partial coverage
- Even with real-time ray tracing, this is definitely not solved.
- When one looks at the images on the right from the Maxwell renderer, clear & rough + shadowed… we still have work to do to reach that quality level in real-time, at 1 or two samples per pixel
- If one considers transparency for particles and volumetric effects, one can use the miss shader to trigger rays that will update volumes, and even do some amount of raymarching in hit shaders right now with DXR, but at what cost?
- Actually most of the volumetric effects require non-trivial blending with the rest of the scene, and non-trivial filtering.
- For transparency on PICA PICA we came up with a texture-space OIT technique, which worked for refractions and scattering
- While our approach worked for our case, even handles view-dependent parameters, it's not perfect and came with its share of temporal issues.
- I'm also not sure how it would blend with other effects we skipped for this demo… such as volumetric effects, particles and fog. One step closer though…
- The thing is, with 1 sample per pixel, and throwing some Monte Carlo in there, most denoising techniques generally don't work so well with transparency or partial coverage



SEED // Game Ray Tracing: State-of-the-Art and Open Problems

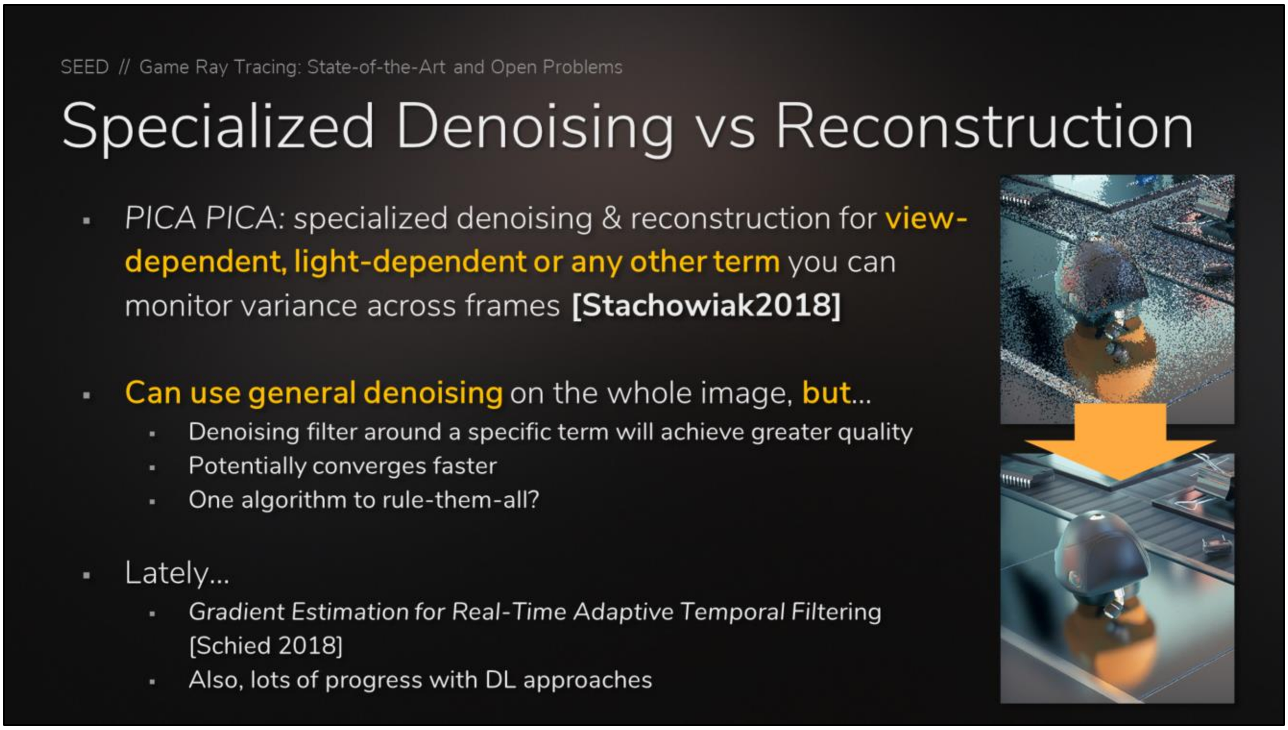
Partial Coverage

- **Foliage**
  - Can still do alpha test
  - Animated becomes a real problem
- **Defocus effects** such as motion blur and depth of field are still intractable for the same reasons
- Need denoising techniques that handle partial coverage @ 1-2 spp ☺

PBRT (Matt Pharr, Wenzel Jakob, Greg Humphreys)

© Disney/Pixar

- Same thing if we talk about foliage and just like the previous fall in the category of partial coverage…
- We can still do alpha testing in the hit shaders, and could use some pre-filtering. But as soon as it starts moving, it becomes a problem both for tracing performance, for BVH updates, but also for filtering.
- The BVH update is technically an IHV issue, but doesn't mean we can just forget about it. ☺
- Refitting is fast, but not free, and if you're building a jungle or a scene like PBRT, or the forest on the island from Moana that Disney has released, it's a different story than just a tree here and there.
- Other types of partial coverage effects that get affected such as depth of field and motion blur also fall into this trap
- And so, current denoising techniques don't work well with this kind of partial visibility in real-time. And often this is because we only have 1 sample per pixel and assume everything is opaque.
- So if we can figure out denoising of partial coverage for 1-2 spp, that would be ace!

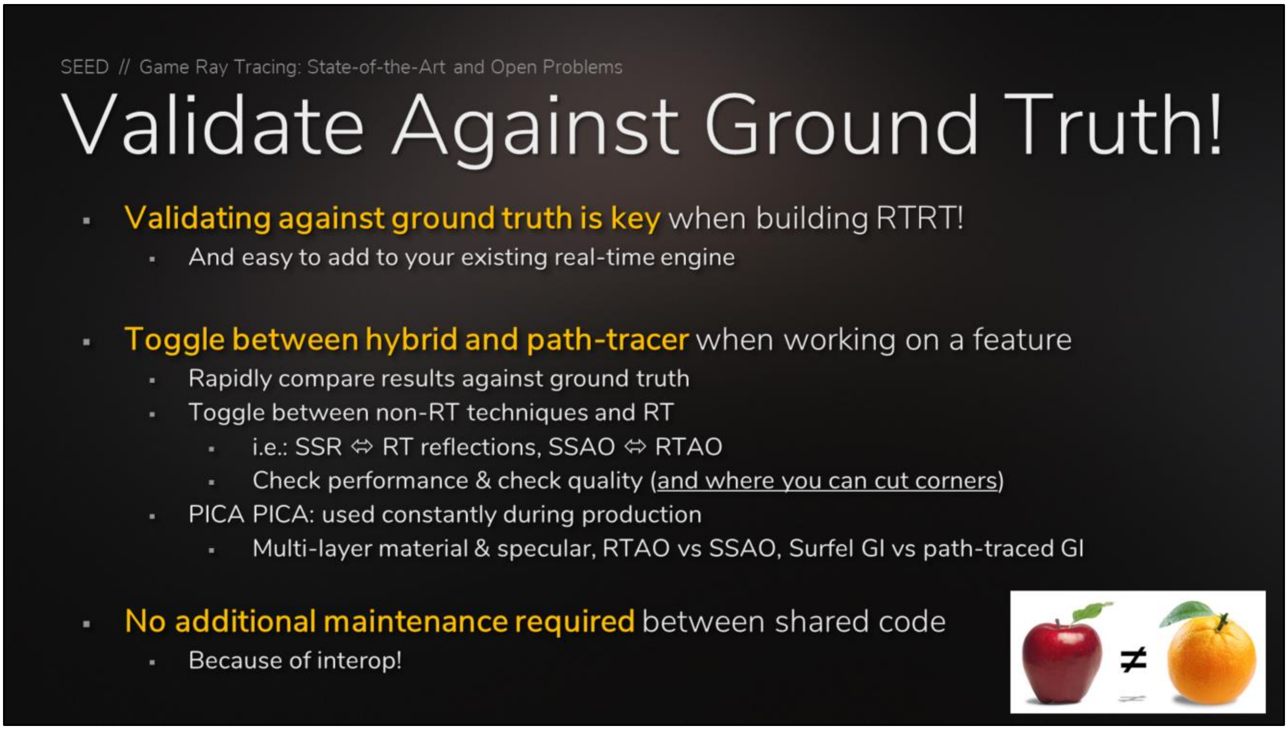


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Specialized Denoising vs Reconstruction

- PICA PICA: specialized denoising & reconstruction for **view-dependent, light-dependent or any other term** you can monitor variance across frames **[Stachowiak2018]**
- **Can use general denoising** on the whole image, **but...**
  - Denoising filter around a specific term will achieve greater quality
  - Potentially converges faster
  - One algorithm to rule-them-all?
- Lately...
  - *Gradient Estimation for Real-Time Adaptive Temporal Filtering* [Schied 2018]
  - Also, lots of progress with DL approaches

- Speaking of denoising and reconstruction, where most games right now support some minimal amount of it, and the fact that there's been great progress there, it feels like we're still not done exploring
- As you saw from our demo, there is definitely something to be said about reusing temporal and spatial data to your advantage
- For example in the case of PICA PICA, we applied denoising and reconstruction that's specialized for view-dependent, lighting dependent or any other term you can monitor variance across frames.
- One _can_ use general denoising on a whole image, but at the end of the day specialized denoisers should always achieved greater quality: you know what problem you are trying to solve, and you focus on that.
- It also should converge faster since it's specialized
- But does that mean we can't find an algorithm to rule them all? Maybe?
- For PICA PICA we used SVGF, and now Christoph Shied has come up with an improved version of SVGF where temporal ghosting is reduced by taking a fraction of samples from the past and checking if they look different in the present, detecting temporal gradients.
- Will need to take a look at it when I go back to Stockholm, looks like really great promising work!
- Also, there's been a lot of progress with deep learning approaches. Seems those networks react well to noise, so might as well throw it at them. Think we're not done here, but definitely interesting to keep an eye on.



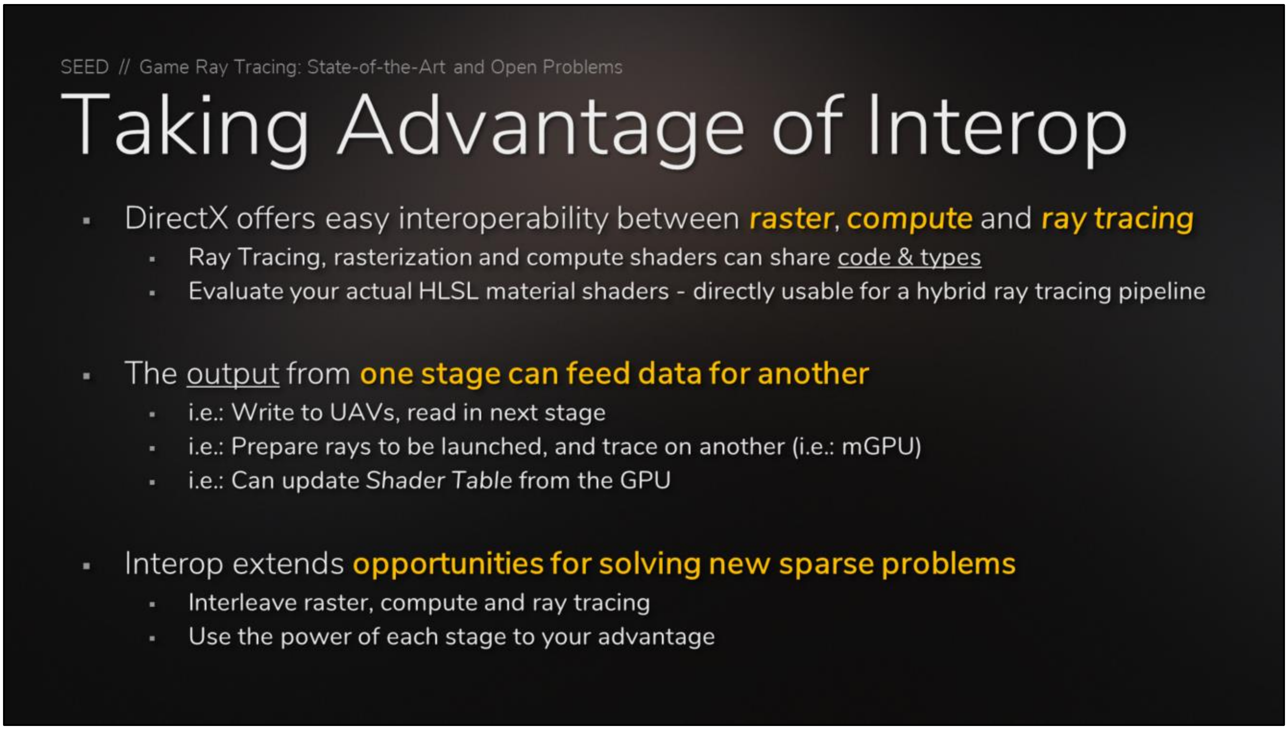
SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Validate Against Ground Truth!

- **Validating against ground truth is key** when building RTRT!
  - And easy to add to your existing real-time engine
- **Toggle between hybrid and path-tracer** when working on a feature
  - Rapidly compare results against ground truth
  - Toggle between non-RT techniques and RT
    - i.e.: SSR ⇔ RT reflections, SSAO ⇔ RTAO
    - Check performance & check quality (and where you can cut corners)
  - PICA PICA: used constantly during production
    - Multi-layer material & specular, RTAO vs SSAO, Surfel GI vs path-traced GI
- **No additional maintenance required** between shared code
  - Because of interop!

- Speaking of implementing techniques, one cool thing I find with DXR is how easy one can adapt their existing real-time engine to have support for ground truth
- As everyone knows here validating against ground truth is key, and necessary when building real-time ray tracing
- For PICA PICA we’ve added the ability to toggle between our hybrid rendering approach and a path tracer implementation
- This has allowed us to rapidly compare results against ground truth, and was super useful as we built our hybrid renderer
- One of the best reasons I find to have this, other than validating that your math is not complete garbage or missing a divide by PI (but then again doesn’t happen often because we use the same HLSL code between the path tracer and the hybrid mode), is to compare and figure out where to cut corners for real-time.
- Figure out when something is good enough and not completely too far from the reference.
- Also because of interop between Raster, Compute and Ray Tracing and like I said the fact that you can share shader code with all pipelines, there is minimal to almost no extra work and maintenance required, so having this ground truth comparison tool shouldn’t add too much work.
- I think this is really cool for researchers, who definitely want to compare against ground truth and swear by it.

- Image: http://www.cristanwilliams.com/b/2012/01/10/apple-separatism/

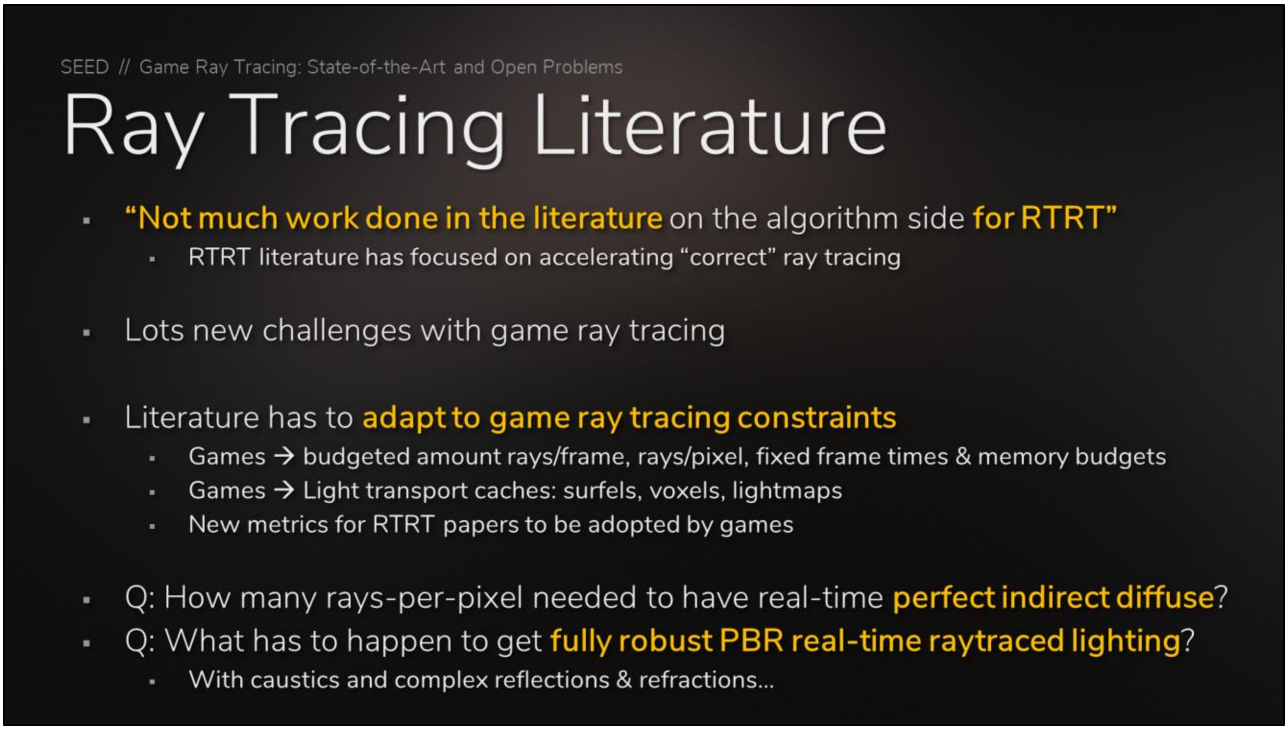


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Taking Advantage of Interop

- DirectX offers easy interoperability between **raster**, **compute** and **ray tracing**
  - Ray Tracing, rasterization and compute shaders can share code & types
  - Evaluate your actual HLSL material shaders - directly usable for a hybrid ray tracing pipeline
- The output from **one stage can feed data for another**
  - i.e.: Write to UAVs, read in next stage
  - i.e.: Prepare rays to be launched, and trace on another (i.e.: mGPU)
  - i.e.: Can update *Shader Table* from the GPU
- Interop extends **opportunities for solving new sparse problems**
  - Interleave raster, compute and ray tracing
  - Use the power of each stage to your advantage

- Speaking of interop, as I mentioned we use the same HLSL code between raster, compute and ray tracing, our hybrid mode and the path tracing reference
- This is really awesome, so no need to do conversions like one had to do in the past when using an offline tool as the reference.
- It always becomes tricky with materials, and so now because it’s the same code, often via generalized functions that you’ve built, makes life easier.
- And so with interop, one stage can feed data for another
- Think of it this way, you can prepare data in compute write it to a UAV and us it in a ray tracing stage. This is what we do transparency and translucency
- You can also prepare rays to be launched, and trace at a later time. This is what we do with our parallel fork-and-join approach mGPU that I talked about
- You can also technically update the shader table from the GPU
- And so, for a researcher, the concept of interop extends opportunities for solving new sparse problems
- You can basically use the power of each stage to your advantage, and develop a technique that can get adopted by a game title.

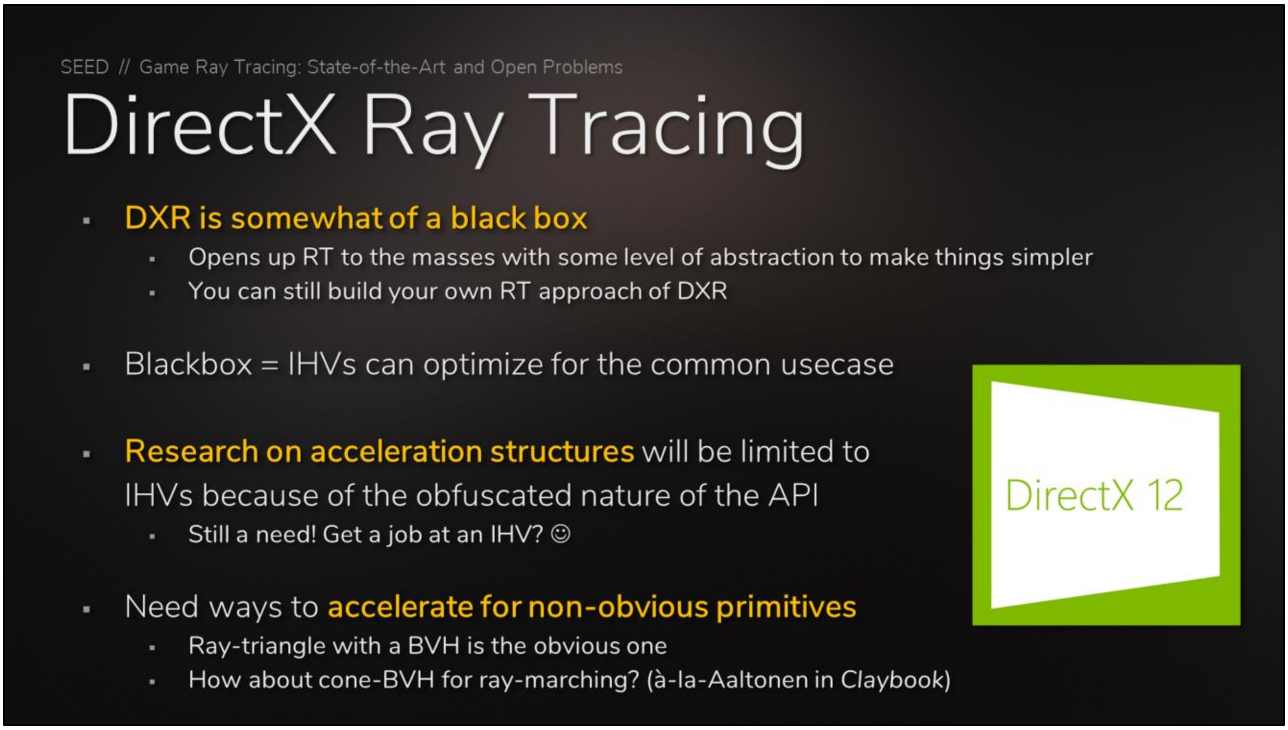


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Ray Tracing Literature

- **"Not much work done in the literature** on the algorithm side **for RTRT"**
  - RTRT literature has focused on accelerating "correct" ray tracing
- Lots new challenges with game ray tracing
- Literature has to **adapt to game ray tracing constraints**
  - Games → budgeted amount rays/frame, rays/pixel, fixed frame times & memory budgets
  - Games → Light transport caches: surfels, voxels, lightmaps
  - New metrics for RTRT papers to be adopted by games
- Q: How many rays-per-pixel needed to have real-time **perfect indirect diffuse**?
- Q: What has to happen to get **fully robust PBR real-time raytraced lighting**?
  - With caustics and complex reflections & refractions...

- If we talk about literature, and hopefully I say this properly and don't offend anyone so please bare with me, and if you have any comments makes sure to share them at the end but…
- While literature has done some significant amount of work to accelerate ray tracing, it has mostly assumed "correct ray tracing". This has undeniably allowed for accelerating BVH building, proposals for hardware acceleration with intersection, shader sorting, etc
- With DXR and game ray tracing this adds a bunch of new challenges, especially if you can only do 1-2-3 or maybe 4 rays per pixel and need to render at either 30 or 60 FPS
- With games people now toying with ray tracing, for publications and wide adoption this means publications have to get adapted to game ray tracing constraints
- Obviously achieving all of these constraints at once is not easy! But it's our reality in the games industry, and we need help from the research community. Maybe not solve everything, since games are flexible, so maybe meet half way?
- Also these new metrics become important for papers to get adopted by games
- Speaking of literature and thinking of game constraints, would love to see someone tackle the following questions
- Not easy! ☺

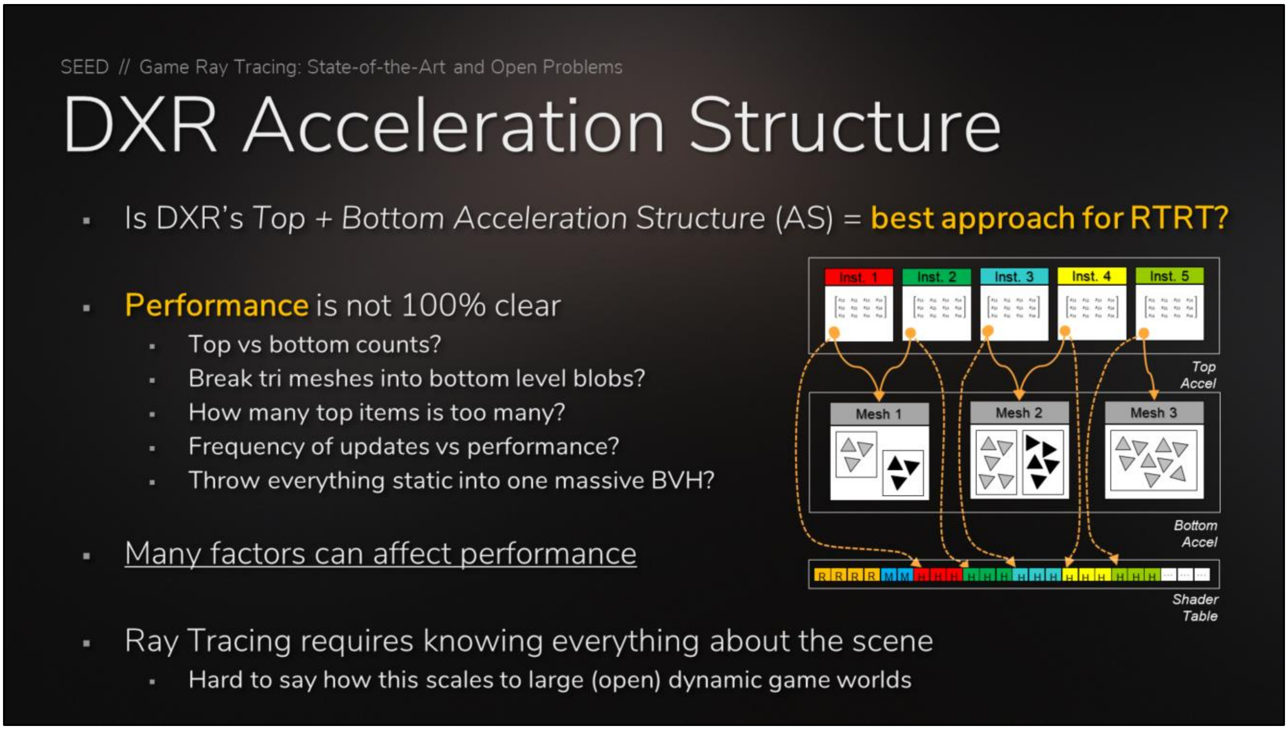


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

DirectX Ray Tracing

- **DXR is somewhat of a black box**
  - Opens up RT to the masses with some level of abstraction to make things simpler
  - You can still build your own RT approach of DXR
- Blackbox = IHVs can optimize for the common usecase
- **Research on acceleration structures** will be limited to IHVs because of the obfuscated nature of the API
  - Still a need! Get a job at an IHV? ☺
- Need ways to **accelerate for non-obvious primitives**
  - Ray-triangle with a BVH is the obvious one
  - How about cone-BVH for ray-marching? (à-la-Aaltonen in *Claybook*)

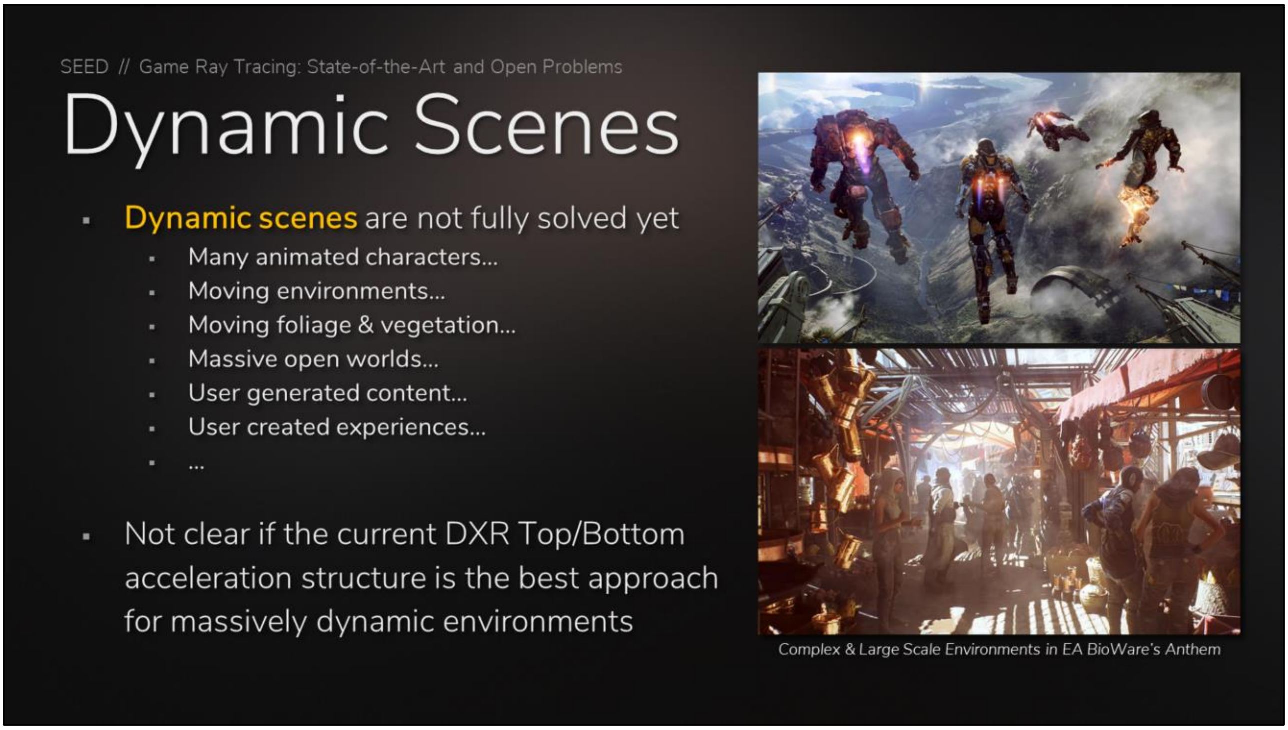
- Speaking of DXR, if you've checked the API, it's a bit of blackbox. Its abstractions is that it makes it available for everyone to speak the same language and have a common playground
- It's quite flexible but has some limitations. It's not forward shading friendly in the sense that you can't launch rays from raster
- Obviously DXR doesn`t prevent you from building your own ray tracing framework with compute
- It being a blackbox, this means IHVs can optimize for the common usecase
- The acceleration structures are opaque, and so optimizations on acceleration structures will be limited to IHVs
- But that research is much needed. I guess one way to have an impact there is to get a job at an IHV? Or experiment with the Fallback layer in Compute and build your own accelerations there…
- We also need to find ways to accelerate for non-obvious primitives. If I were to build some ray tracing hardware, ray-triangle in a BVH would the obvious one
- But cone-BVH might also be interesting, especially for ray marching



DXR Acceleration Structure

- Is DXR's *Top + Bottom Acceleration Structure* (AS) = **best approach for RTRT?**
- **Performance** is not 100% clear
  - Top vs bottom counts?
  - Break tri meshes into bottom level blobs?
  - How many top items is too many?
  - Frequency of updates vs performance?
  - Throw everything static into one massive BVH?
- Many factors can affect performance
- Ray Tracing requires knowing everything about the scene
  - Hard to say how this scales to large (open) dynamic game worlds

- And so earlier I mentioned the Top and Bottom acceleration structures for DXR, and whether that's the best approach for real-time ray tracing and dynamic scenes
- Right now it's not clear how performance scales in the context of AAA game titles
- Considering you can have many-to-1, what about top vs bottom counts
- Or other questions such as the ones listed here
- Is the best approach to throw everything into a single massive BVH?
- Many factors can affect the performance, so it's not clear yet. There aren't a lot of docs going into detail about expected counts and usage characteristics at the moment.
- Also, since ray tracing requires knowing everything about the scene, it's kind hard right now to know how it scales to massive dynamic open 3x3, 5x5, 10x10 km worlds



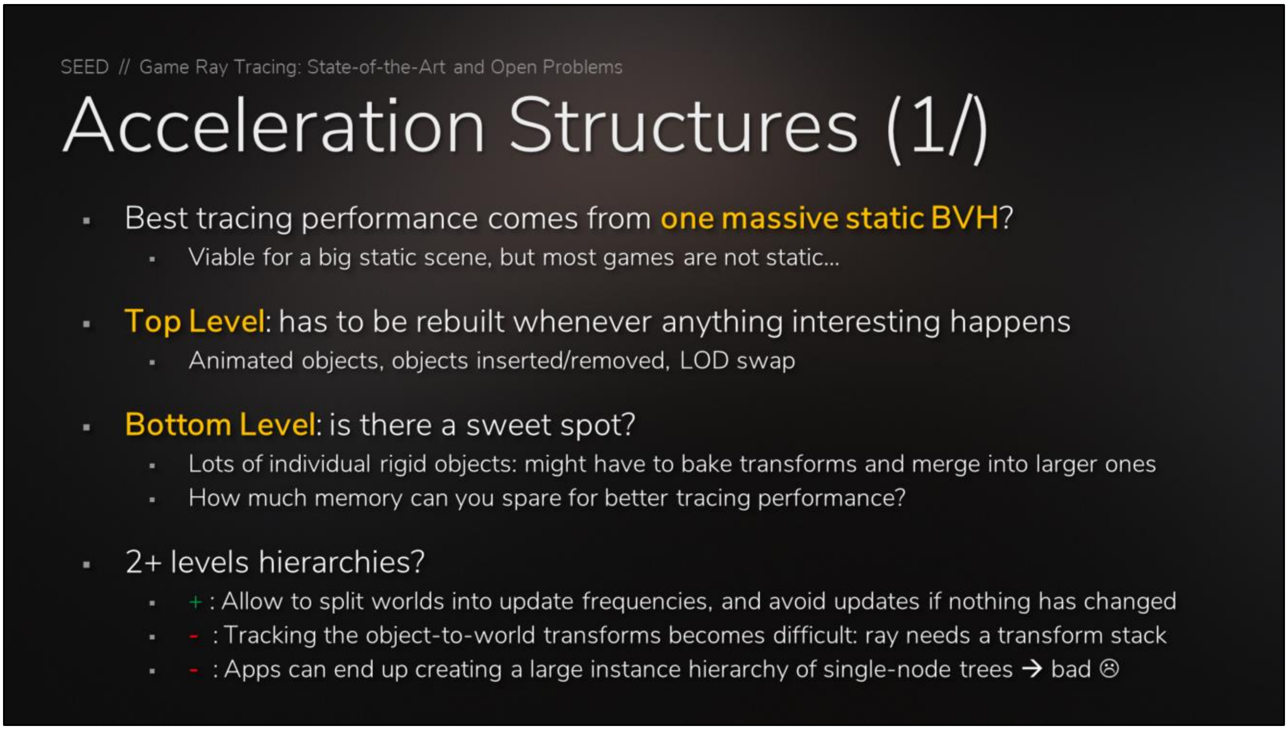
SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Dynamic Scenes

- **Dynamic scenes** are not fully solved yet
  - Many animated characters…
  - Moving environments…
  - Moving foliage & vegetation…
  - Massive open worlds…
  - User generated content…
  - User created experiences…
  - …
- Not clear if the current DXR Top/Bottom acceleration structure is the best approach for massively dynamic environments

Complex & Large Scale Environments in EA BioWare's Anthem

- Speaking of massive worlds, this also means dynamic scenes, and dynamic scenes are not fully solved yet in terms of performance
- In the context of a game we need to support many animated characters, foliage, potentially in a massive open world that evolves
- With some user generated content and created experiences that you might not able to process on the fly, without any art clean-up
- And so in terms of dynamism, it's not clear if the current top & bottom acceleration structure is the best approach for massive dynamic environments

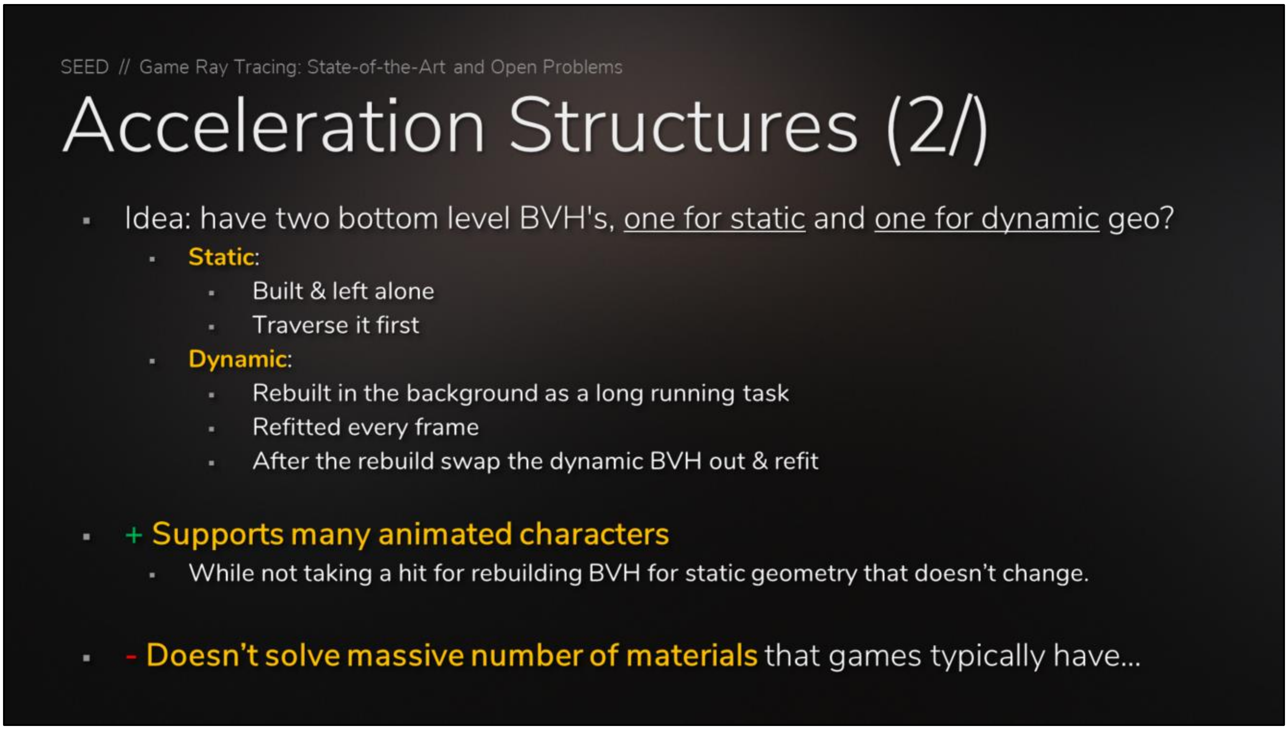


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Acceleration Structures (1/)

- Best tracing performance comes from **one massive static BVH**?
  - Viable for a big static scene, but most games are not static…
- **Top Level**: has to be rebuilt whenever anything interesting happens
  - Animated objects, objects inserted/removed, LOD swap
- **Bottom Level**: is there a sweet spot?
  - Lots of individual rigid objects: might have to bake transforms and merge into larger ones
  - How much memory can you spare for better tracing performance?
- 2+ levels hierarchies?
  - + : Allow to split worlds into update frequencies, and avoid updates if nothing has changed
  - - : Tracking the object-to-world transforms becomes difficult: ray needs a transform stack
  - - : Apps can end up creating a large instance hierarchy of single-node trees → bad ☹

- We’ve been told to shove everything in one BVH right now. And to that one question one might have is whether having a single BVH is the best way to get the best performance
- This feels viable for a static scene, but most games are not static
- And so the top level has to be rebuilt when anything interesting happens, such as when objects move but also when static geometry LOD gets swapped are you get closer to a building.
- With regards to the bottom level acceleration, it’s not clear what the sweet spot is: one can have lots of individual rigid objects, which will take memory, and so to fix this we might have to merge down objects.
- Not sure what the numbers are yet, we have to experiment more because of the blackbox nature of the API.
- And so what about more than 2 level hierarchies?
- This would allow to split worlds in update frequencies
- But also means that tracking object-to-object transformations becomes a stack which makes it more complicated for ISVs
- Also IHVs might fear that aps can end up creating large instance hierarchies of single node trees. This would be bad for performance.

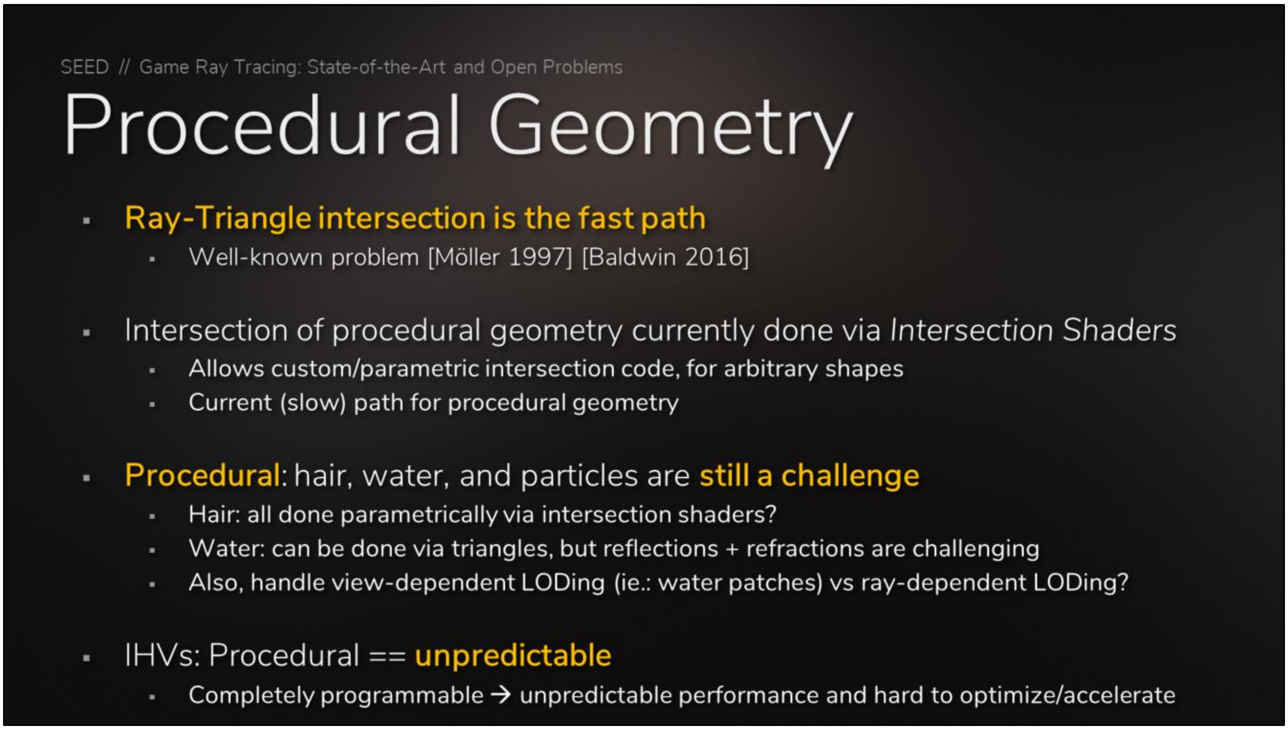


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Acceleration Structures (2/)

- Idea: have two bottom level BVH's, one for static and one for dynamic geo?
  - **Static**:
    - Built & left alone
    - Traverse it first
  - **Dynamic**:
    - Rebuilt in the background as a long running task
    - Refitted every frame
    - After the rebuild swap the dynamic BVH out & refit
- + **Supports many animated characters**
  - While not taking a hit for rebuilding BVH for static geometry that doesn't change.
- - **Doesn't solve massive number of materials** that games typically have...

- Here's a suggestion: have two bottom level BVHs, one for static and one for dynamic geometry
- And so the static one is built once and left alone. This is the one you traverse first
- The tynamic one is rebuilt in the background as a long runniing tasks, refitted every frame, and if you have to do a rebuild you can have 2 copies that can be swapped when rebuilding is done
- This approach would support many animated characters easily while not taking a hit for rebuilding the BVH for static geometry which doesn`t change
- Doesn`t solve the issue of massive number of materials that typical game engines have
- Just an idea, and we wonder if you have other ideas on how to solve this. If you have something to share, would love to hear it at the end during the questions

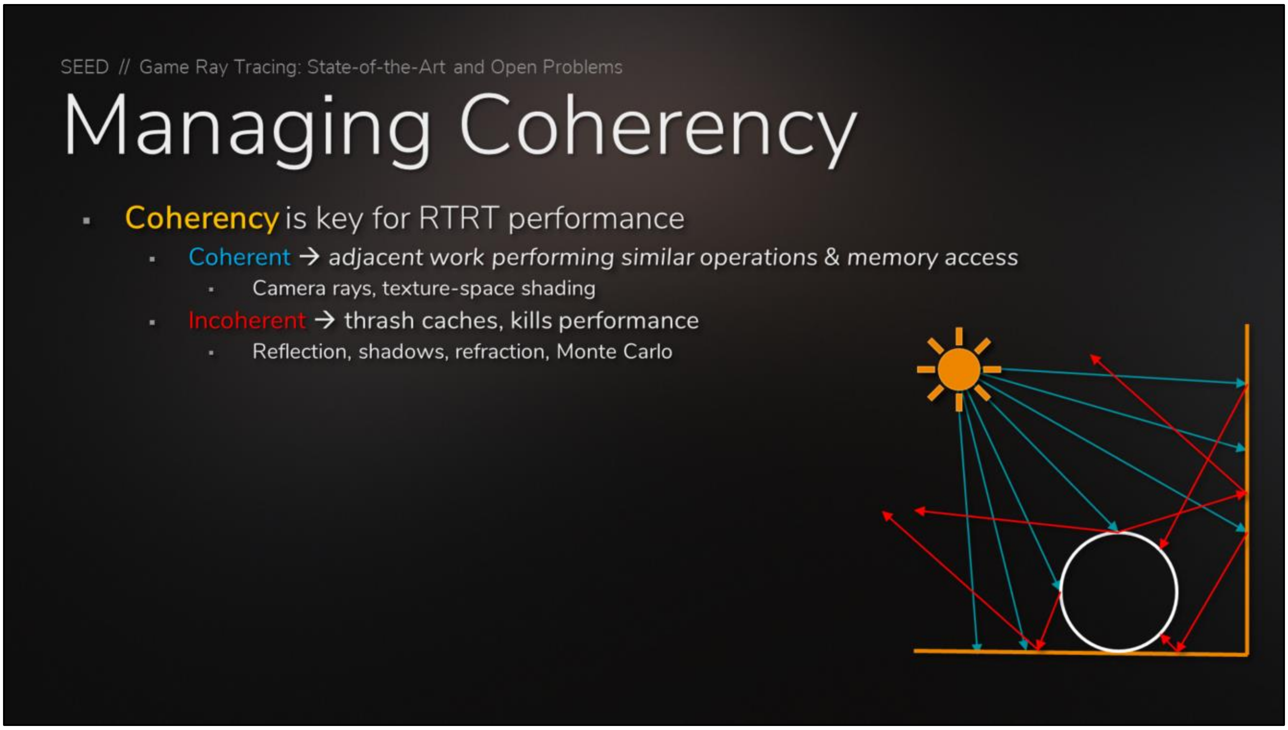


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Procedural Geometry

- **Ray-Triangle intersection is the fast path**
  - Well-known problem [Möller 1997] [Baldwin 2016]
- Intersection of procedural geometry currently done via *Intersection Shaders*
  - Allows custom/parametric intersection code, for arbitrary shapes
  - Current (slow) path for procedural geometry
- **Procedural**: hair, water, and particles are **still a challenge**
  - Hair: all done parametrically via intersection shaders?
  - Water: can be done via triangles, but reflections + refractions are challenging
  - Also, handle view-dependent LODing (ie.: water patches) vs ray-dependent LODing?
- IHVs: Procedural == **unpredictable**
  - Completely programmable → unpredictable performance and hard to optimize/accelerate

- When it comes to geometry, ray-triangle intersection is the fast path. This is a well known problem, and supported natively by the API and the current GPU native implementations
- On the flipside, in DXR intersection of procedural geometry is done via intersection shaders which allows for custom intersection code, basically arbitrary shapess
- This is obviously the slow path. With the current drivers I calculated calculated that procedural triangle intersection is about half as fast at the native one
- This means that anything procedural, such as hair water and particles is still a challenge
- In the case of water, you can use triangles,. But reflections and refractions are challenging
- And so in the IHV perspective, right now procedural means unpredictable since you can do pretty much whatever you want
- This is hard to optimize and unpredictable performance-wise

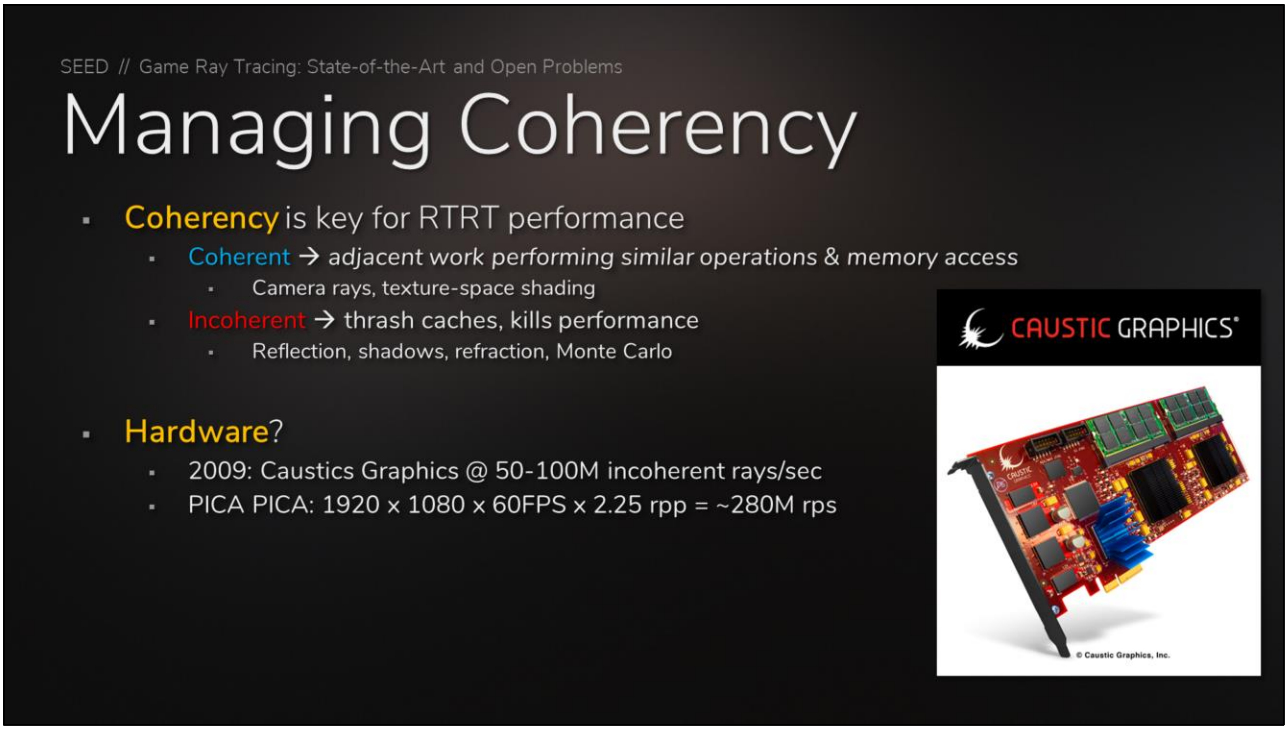


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Managing Coherency

- Coherency is key for RTRT performance
  - Coherent → *adjacent work performing similar operations & memory access*
    - Camera rays, texture-space shading
  - Incoherent → thrash caches, kills performance
    - Reflection, shadows, refraction, Monte Carlo

- Speaking of performance, handling coherency is key for real-time ray tracing performance
- You will get some adjacent rays that perform similar operations and memory accesses, and those will perform well, while some might trash cache and affect performance
- Depending on what techniques you implement, you will have to keep this in mind
- And while there have been attempts at ray tracing hardware in the past, it doesn't mean that it will solve everything
- In the content of pica pica, if all rays were incoherent we'd need hardware that can do 280M incoherent rays per second
- Can't expect out-of-core ray sorting and coherency construction from total mess. Still need to tackle coherency upfront in the techniques & algorithms we develop.
- There has been some work on the software side, especially from Disney and their Hyperion renderer
- But more R&D and pipeline tailoring will be needed to reduce incoherencies for real-time



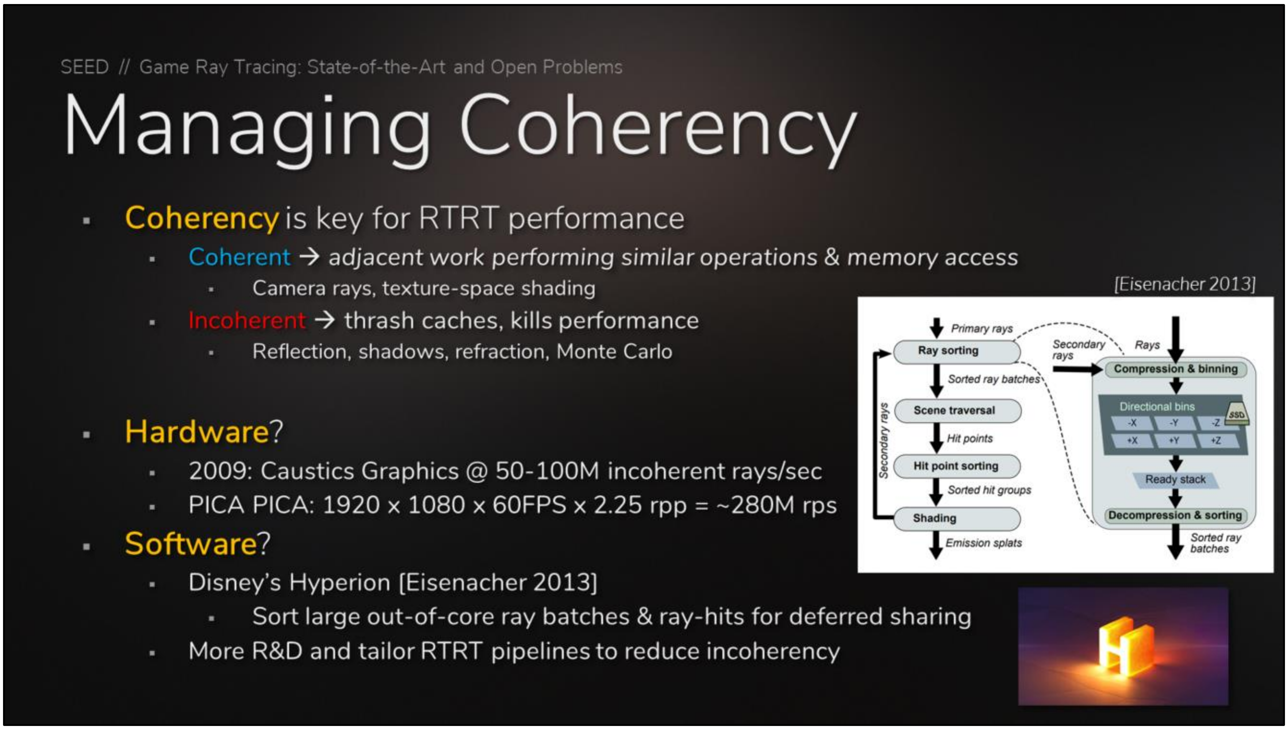
SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Managing Coherency

- Coherency is key for RTRT performance
  - Coherent → *adjacent work performing similar operations & memory access*
    - Camera rays, texture-space shading
  - Incoherent → thrash caches, kills performance
    - Reflection, shadows, refraction, Monte Carlo
- Hardware?
  - 2009: Caustics Graphics @ 50-100M incoherent rays/sec
  - PICA PICA: 1920 x 1080 x 60FPS x 2.25 rpp = ~280M rps

© Caustic Graphics, Inc.

- Speaking of performance, handling coherency is key for real-time ray tracing performance
- You will get some adjacent rays that perform similar operations and memory accesses, and those will perform well, while some might trash cache and affect performance
- Depending on what techniques you implement, you will have to keep this in mind
- And while there have been attempts at ray tracing hardware in the past, it doesn’t mean that it will solve everything
- In the content of pica pica, if all rays were incoherent we’d need hardware that can do 280M incoherent rays per second
- Can’t expect out-of-core ray sorting and coherency construction from total mess. Still need to tackle coherency upfront in the techniques & algorithms we develop.
- There has been some work on the software side, especially from Disney and their Hyperion renderer
- But more R&D and pipeline tailoring will be needed to reduce incoherencies for real-time



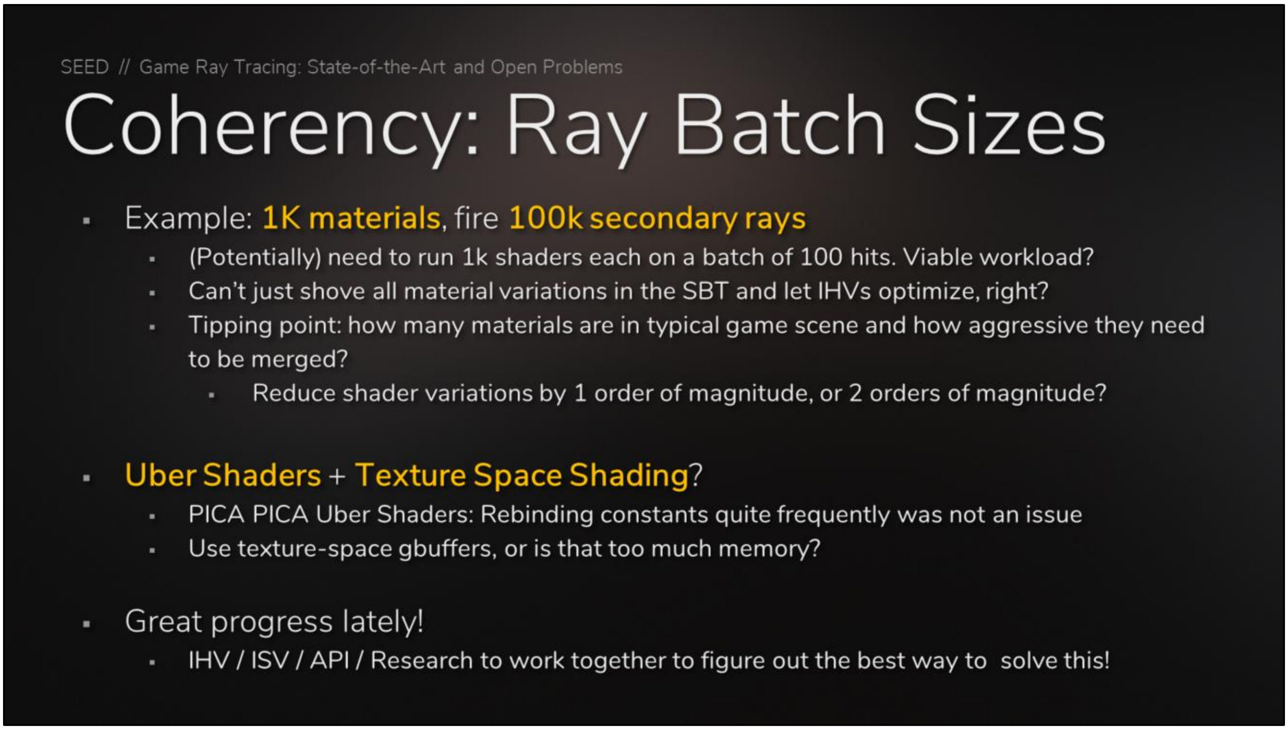
SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Managing Coherency

- Coherency is key for RTRT performance
  - Coherent → *adjacent work performing similar operations & memory access*
    - Camera rays, texture-space shading
  - Incoherent → thrash caches, kills performance
    - Reflection, shadows, refraction, Monte Carlo
- Hardware?
  - 2009: Caustics Graphics @ 50-100M incoherent rays/sec
  - PICA PICA: 1920 x 1080 x 60FPS x 2.25 rpp = ~280M rps
- Software?
  - Disney's Hyperion [Eisenacher 2013]
    - Sort large out-of-core ray batches & ray-hits for deferred sharing
  - More R&D and tailor RTRT pipelines to reduce incoherency

[Eisenacher 2013]

- Speaking of performance, handling coherency is key for real-time ray tracing performance
- You will get some adjacent rays that perform similar operations and memory accesses, and those will perform well, while some might trash cache and affect performance
- Depending on what techniques you implement, you will have to keep this in mind
- And while there have been attempts at ray tracing hardware in the past, it doesn't mean that it will solve everything
- In the content of pica pica, if all rays were incoherent we'd need hardware that can do 280M incoherent rays per second
- Can't expect out-of-core ray sorting and coherency construction from total mess. Still need to tackle coherency upfront in the techniques & algorithms we develop.
- There has been some work on the software side, especially from Disney and their Hyperion renderer
- But more R&D and pipeline tailoring will be needed to reduce incoherencies for real-time

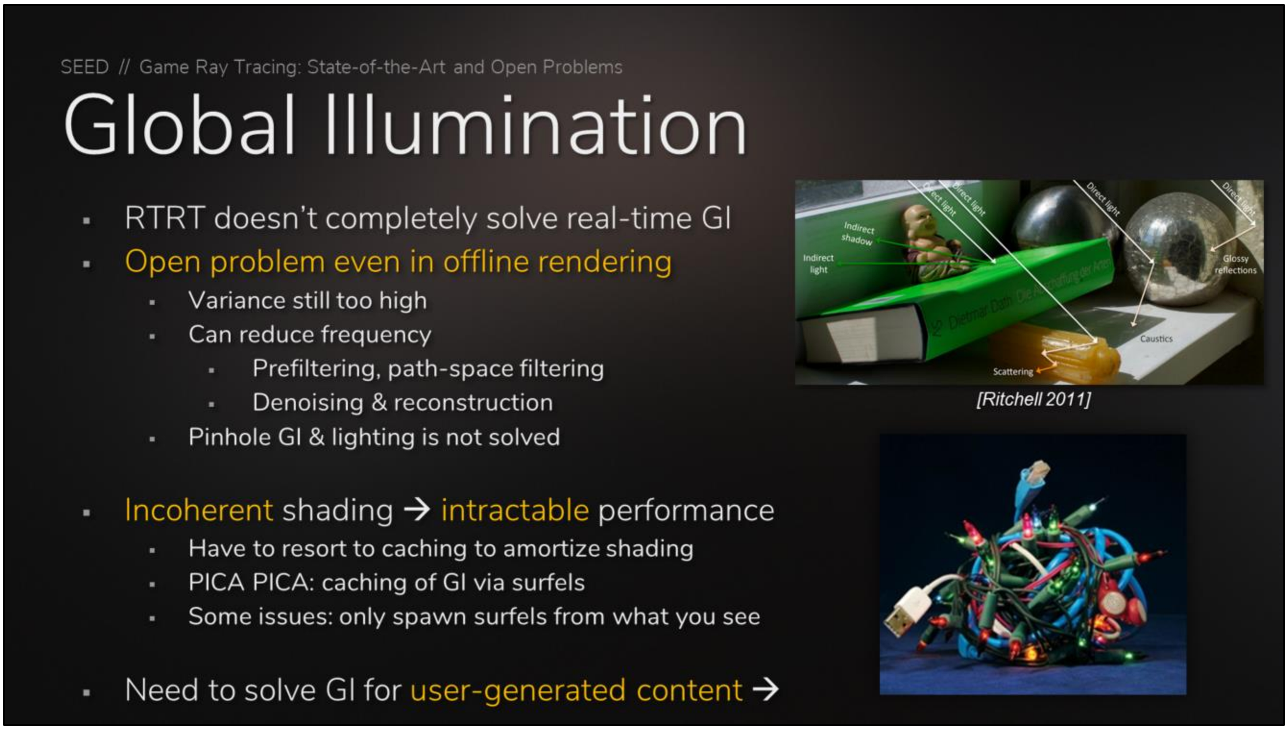


SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Coherency: Ray Batch Sizes

- Example: **1K materials**, fire **100k secondary rays**
  - (Potentially) need to run 1k shaders each on a batch of 100 hits. Viable workload?
  - Can't just shove all material variations in the SBT and let IHVs optimize, right?
  - Tipping point: how many materials are in typical game scene and how aggressive they need to be merged?
    - Reduce shader variations by 1 order of magnitude, or 2 orders of magnitude?
- **Uber Shaders** + **Texture Space Shading**?
  - PICA PICA Uber Shaders: Rebinding constants quite frequently was not an issue
  - Use texture-space gbuffers, or is that too much memory?
- Great progress lately!
  - IHV / ISV / API / Research to work together to figure out the best way to solve this!

- Managing coherency also means managing ray batch sizes
- For example, in a real game scene, where you have a 1000 materials and need to fire 100 000 secondary rays
- This means that 100 shaders protentially need to run on hundreds of hits. That's a questionably-viable GPU workload
- And so we can't just shove all material variations in the shader binding table and let IHVs figure it out
- There is a tipping point wherewe might have to reduce the number of shader variations, but is it 1 order or 2 orders of magnitude
- Or one might consider adopting more a uber-shader-like approach, like we did for pica pica, but results might vary depending on the kind of game you are making
- Should games move even more to uber shaders (or precomputed shader graphs) and texture-space g-buffers to remove control divergence. That might require too much memory but might work in some cases
- At the end of the day, dumping this responsibility on IHVs is not the solution, but we'll have to work together on what is the best way to provide hints at IHVs for scheduling unoptimal workloads



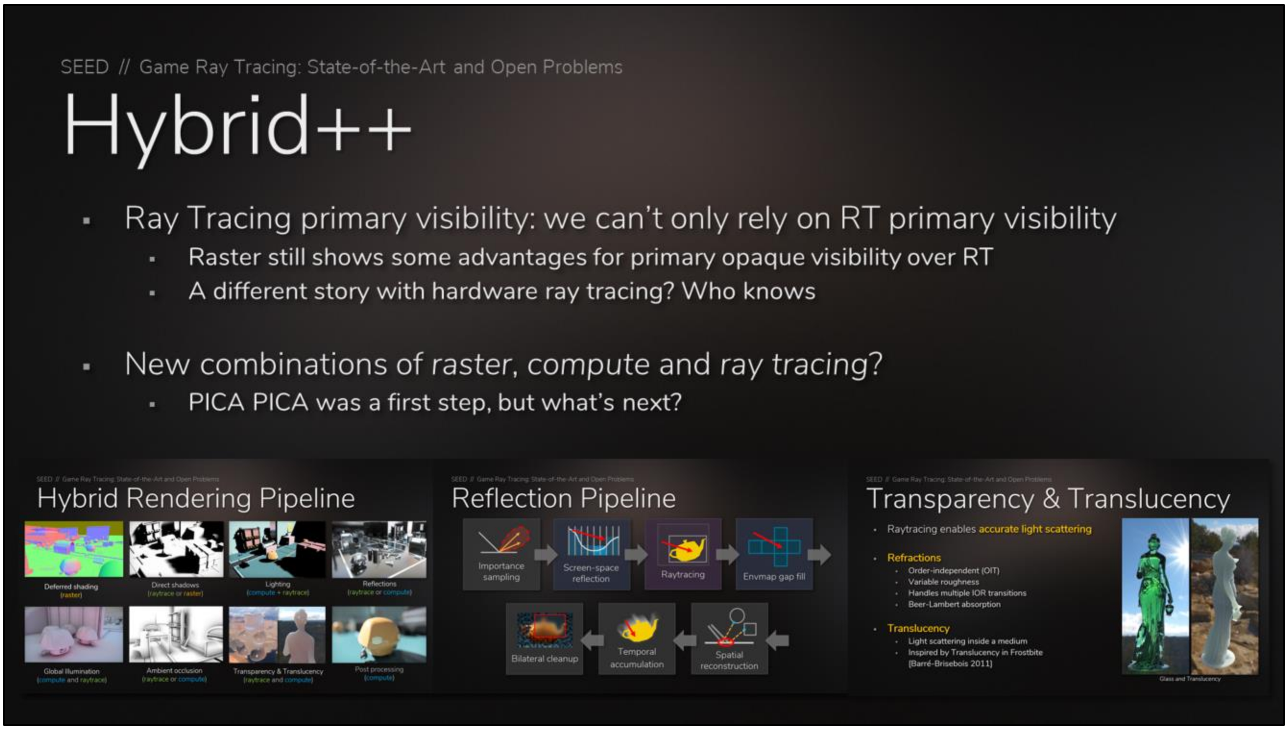
SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Global Illumination

- RTRT doesn't completely solve real-time GI
- Open problem even in offline rendering
  - Variance still too high
  - Can reduce frequency
    - Prefiltering, path-space filtering
    - Denoising & reconstruction
  - Pinhole GI & lighting is not solved
- Incoherent shading → intractable performance
  - Have to resort to caching to amortize shading
  - PICA PICA: caching of GI via surfels
  - Some issues: only spawn surfels from what you see
- Need to solve GI for user-generated content →

[Ritchell 2011]

- Real-time ray tracing doesn`t completely solve real-time GI
- As most of you know, it's still a problem in offline rendering where scenes can take hours to resolve. With difficult paths with caustics for example
- There are many workarounds in offline, but they don't necessarily map to the real-time world
- For PICA PICA we had to resort to caching of GI in surfels. Works but has some limitations. Our artists were happy to not have to deal with lightmap Uvs and mesh proxies though…
- We also need to solve GI for user-generated content, where you can't expect any upfront parametrization
- So even with RTRT, we're definitely not done here



SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Hybrid++

- Ray Tracing primary visibility: we can't only rely on RT primary visibility
  - Raster still shows some advantages for primary opaque visibility over RT
  - A different story with hardware ray tracing? Who knows
- New combinations of *raster, compute and ray tracing?*
  - PICA PICA was a first step, but what's next?

SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Hybrid Rendering Pipeline

SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Reflection Pipeline

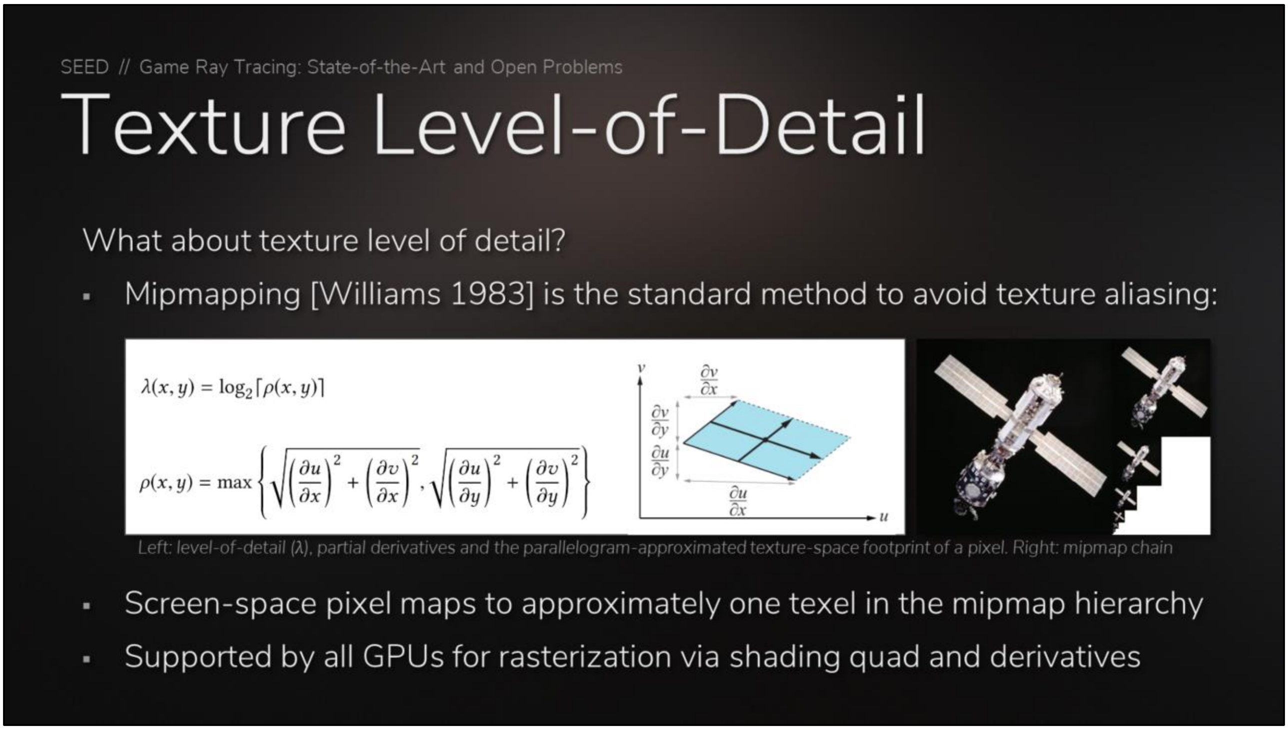
SEED // Game Ray Tracing: State-of-the-Art and Open Problems

Transparency & Translucency

- Raytracing enables **accurate light scattering**
- **Refractions**
  - Order-independent (OIT)
  - Variable roughness
  - Handles multiple IOR transitions
  - Beer-Lambert absorption
- **Translucency**
  - Light scattering inside a medium
  - Inspired by Translucency in Frostbite [Barré-Brisebois 2011]

Glass and Translucency

- At this point in time we can`t rely 100% on raytraced primary visibility. It works, we have prototyped it, but it would be a bit silly to not take advantage of raster, which has some advantages
- If we were to have hardware ray tracing, would probably be a different story, but who knows
- And so we're thinking of other ways of combining raster, compute and ray tracing
- Pica pica was a first step, but what's next. If folks here have ideas, I'm definitely open to chatting more it with you, some of the challenges we had but also where we can take things next



Texture Level-of-Detail

What about texture level of detail?

- Mipmapping [Williams 1983] is the standard method to avoid texture aliasing:

$$\lambda(x,y) = \log_2\lceil \rho(x,y) \rceil$$

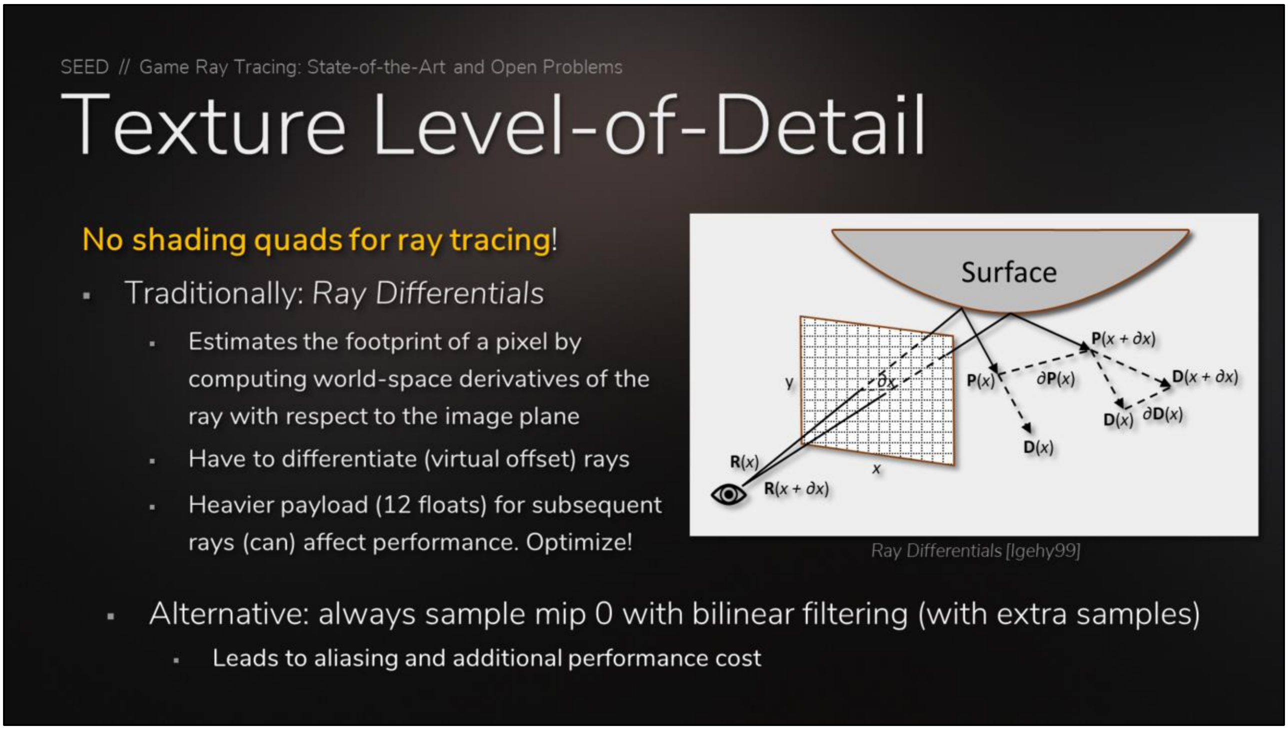
$$\rho(x,y) = \max\left\{\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2}\right\}$$

v
$\frac{\partial v}{\partial x}$
$\frac{\partial v}{\partial y}$
$\frac{\partial u}{\partial y}$
$\frac{\partial u}{\partial x}$
u

Left: level-of-detail (λ), partial derivatives and the parallelogram-approximated texture-space footprint of a pixel. Right: mipmap chain

- Screen-space pixel maps to approximately one texel in the mipmap hierarchy
- Supported by all GPUs for rasterization via shading quad and derivatives

What do I mean about texture LOD?



Texture Level-of-Detail

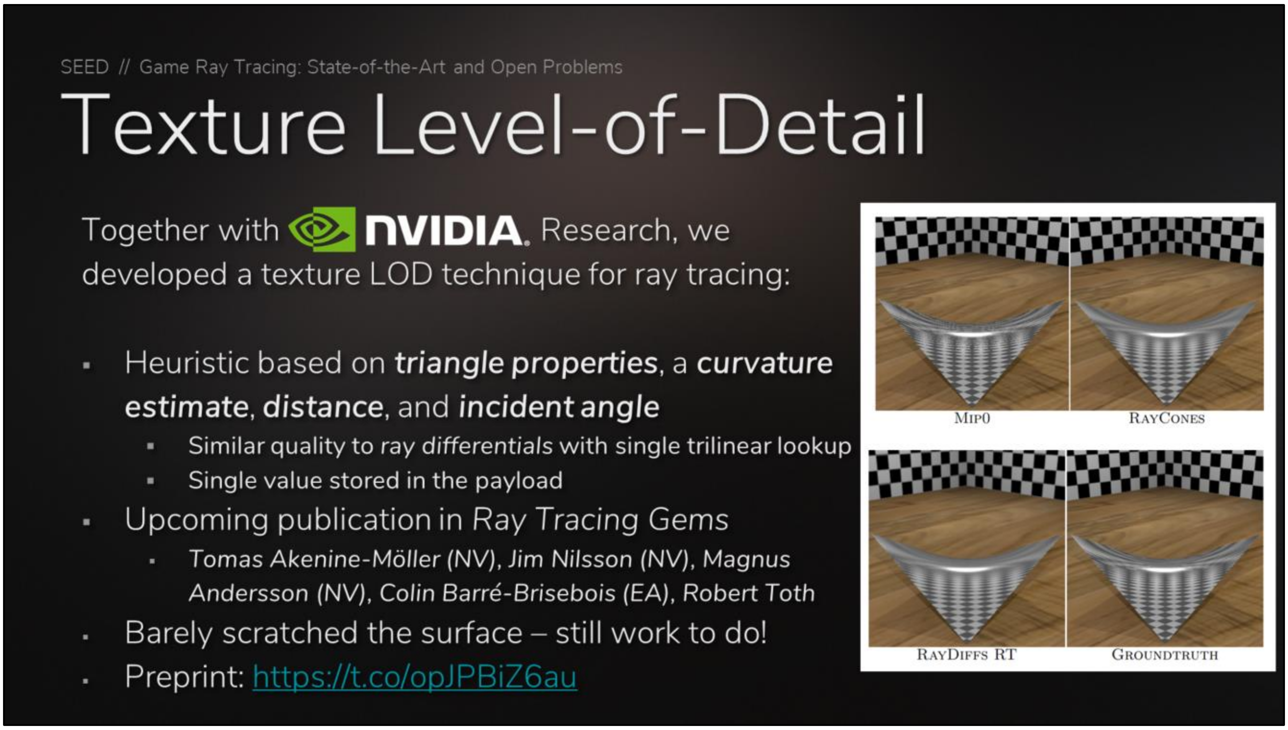
No shading quads for ray tracing!

- Traditionally: *Ray Differentials*
  - Estimates the footprint of a pixel by computing world-space derivatives of the ray with respect to the image plane
  - Have to differentiate (virtual offset) rays
  - Heavier payload (12 floats) for subsequent rays (can) affect performance. Optimize!

Ray Differentials [Igehy99]

- Alternative: always sample mip 0 with bilinear filtering (with extra samples)
  - Leads to aliasing and additional performance cost

- As many of you are aware, we don’t have shading quads for ray tracing

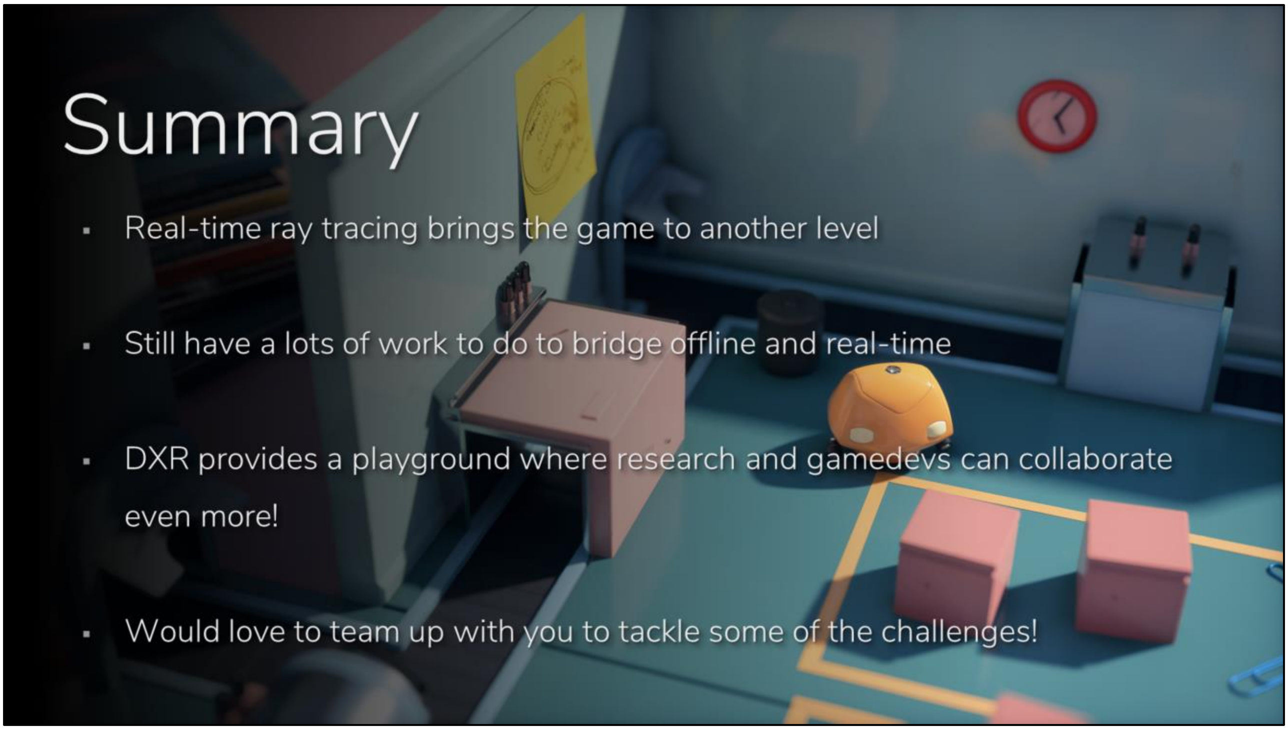


Texture Level-of-Detail

Together with NVIDIA. Research, we developed a texture LOD technique for ray tracing:

- Heuristic based on **triangle properties**, a **curvature estimate**, **distance**, and **incident angle**
  - Similar quality to *ray differentials* with single trilinear lookup
  - Single value stored in the payload
- Upcoming publication in *Ray Tracing Gems*
  - *Tomas Akenine-Möller (NV), Jim Nilsson (NV), Magnus Andersson (NV), Colin Barré-Brisebois (EA), Robert Toth*
- Barely scratched the surface – still work to do!
- Preprint: https://t.co/opJPBiZ6au

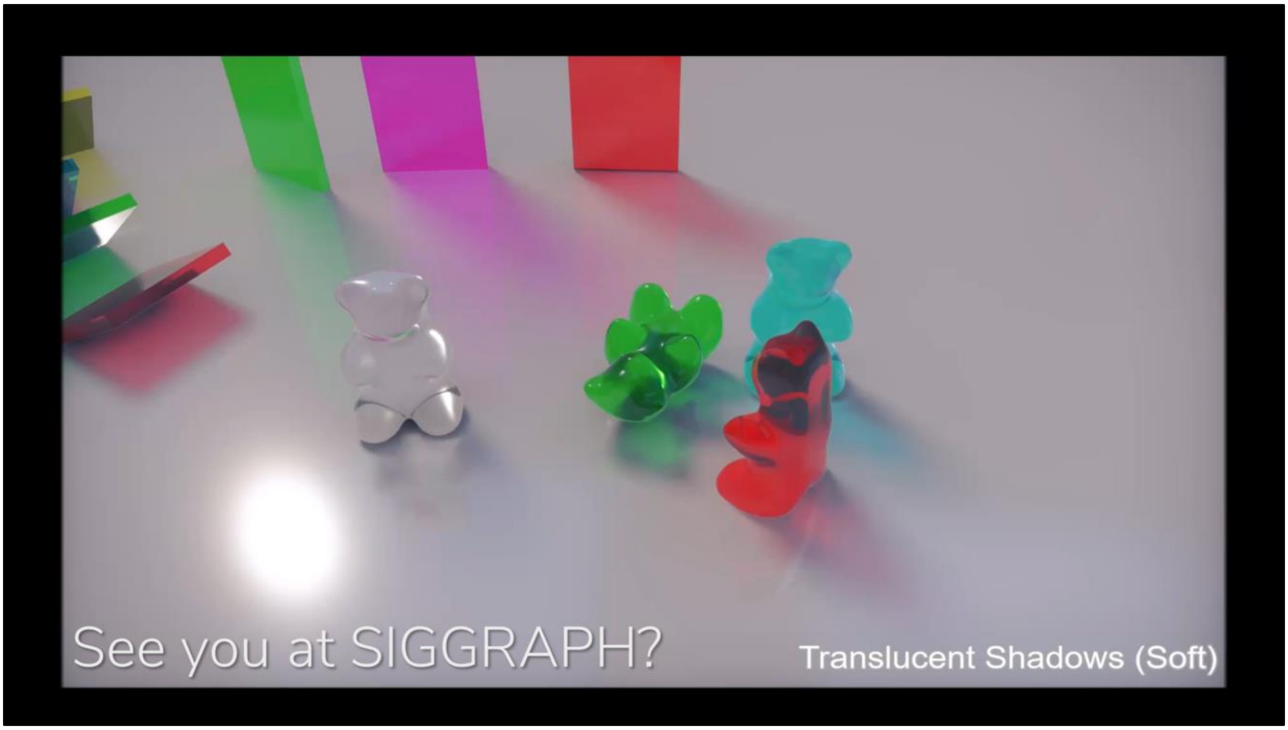
- We’ve barely scratched the surface:
- Works great for materials where the texture sampling is very clear
- Need to look at anisotropy & dependent texture lookups in complex shader graphs
- Technique assumes static Uvs, so need to look at recomputing texture coordinates on the fly?
- For pica pica we used it but the difference is minimal because of the super clean visuals
- Even with detail normal on all objects, with a ton of TAA on top sampling mip0 was not noisy
- Does affect performance, so you don’t want to only sample mip0
- A good start though, but still work to do here. Make sure to check out the pre-print



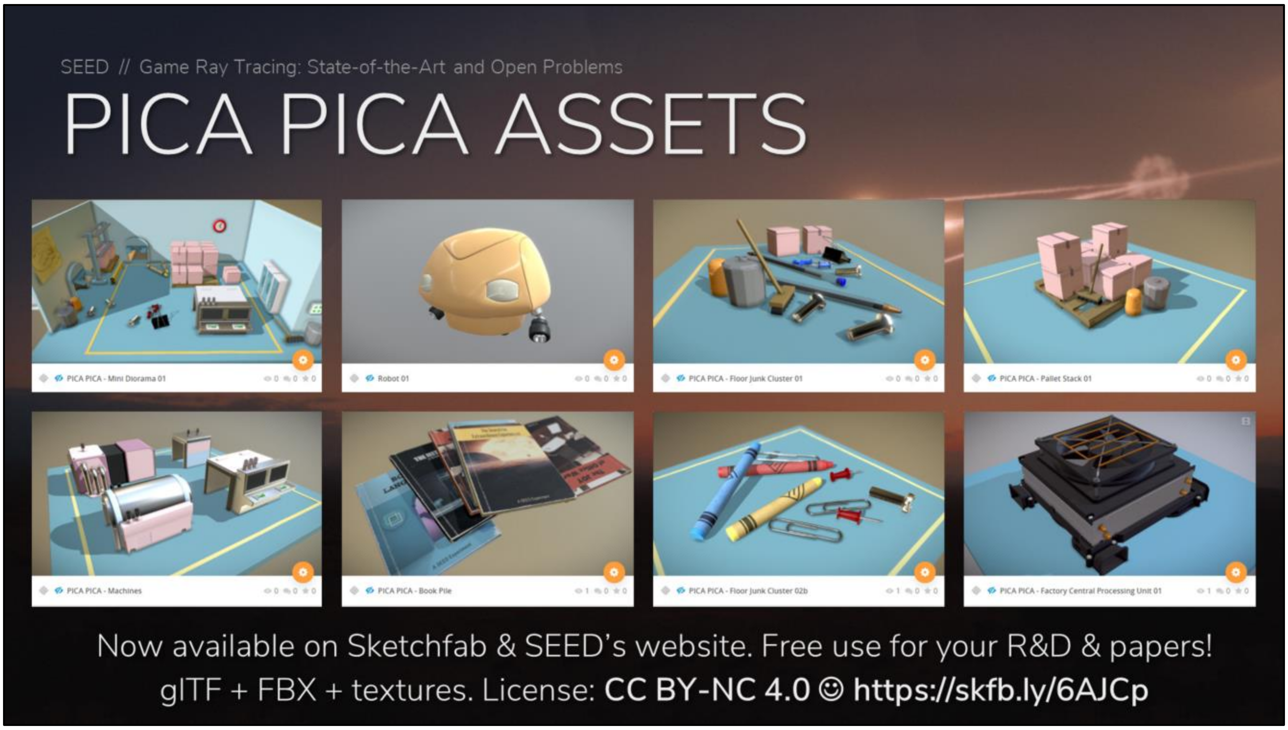
Summary

- Real-time ray tracing brings the game to another level
- Still have a lots of work to do to bridge offline and real-time
- DXR provides a playground where research and gamedevs can collaborate even more!
- Would love to team up with you to tackle some of the challenges!

- That’s what’s fun about graphics, every year with progress from IHVs, ISVs, and research, it gets even more awesome. Looking forward to what we can solve as an industry and where we take things next, together!
- Ping us at SEED if you have some ideas and would like to collaborate.



See you at the NVIDIA real-time ray tracing session where we will talk about additional ray tracing experiments we did, such as soft transparent shadows for glass rendering.



PICA PICA ASSETS

PICA PICA - Mini Diorama 01
Robot 01
PICA PICA - Floor Junk Cluster 01
PICA PICA - Pallet Stack 01
PICA PICA - Machines
PICA PICA - Book Pile
PICA PICA - Floor Junk Cluster 02b
PICA PICA - Factory Central Processing Unit 01

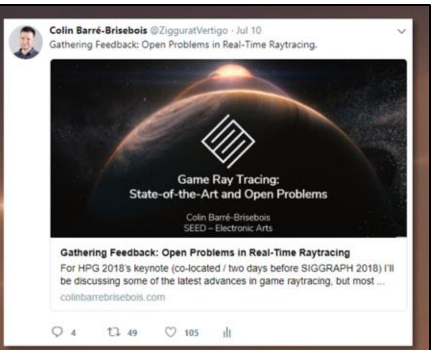
Now available on Sketchfab & SEED’s website. Free use for your R&D & papers!
glTF + FBX + textures. License: **CC BY-NC 4.0 ☺ https://skfb.ly/6AJCp**

- Academia always asks for content from the games people and often doesn’t get it.
- And so for SIGGRAPH we have decided to release all the assets from PICA PICA
- You can download them via Sketchfab and on our website, use them in your research for free, build your ray tracing pipeline and compare with ours.
- Challenge Accepted?

Thanks

SEED PICA PICA Team

- Tomas Akenine-Möller
- Sebastian Aaltonen
- Johan Andersson
- Joshua Barczak
- Jasper Bekkers
- Carlos Gonzalez-Ochoa
- Jon Greenberg
- Henrik Halén
- John Hable
- Steve Hill
- Andrew Lauritzen
- Aaron Lefohn
- Krzysztof Narkowicz
- Richard Schreyer
- Tomasz Stachowiak
- Graham Wihlidal



Colin Barré-Brisebois @ZigguratVertigo · Jul 10

Gathering Feedback: Open Problems in Real-Time Raytracing.

Game Ray Tracing:
State-of-the-Art and Open Problems

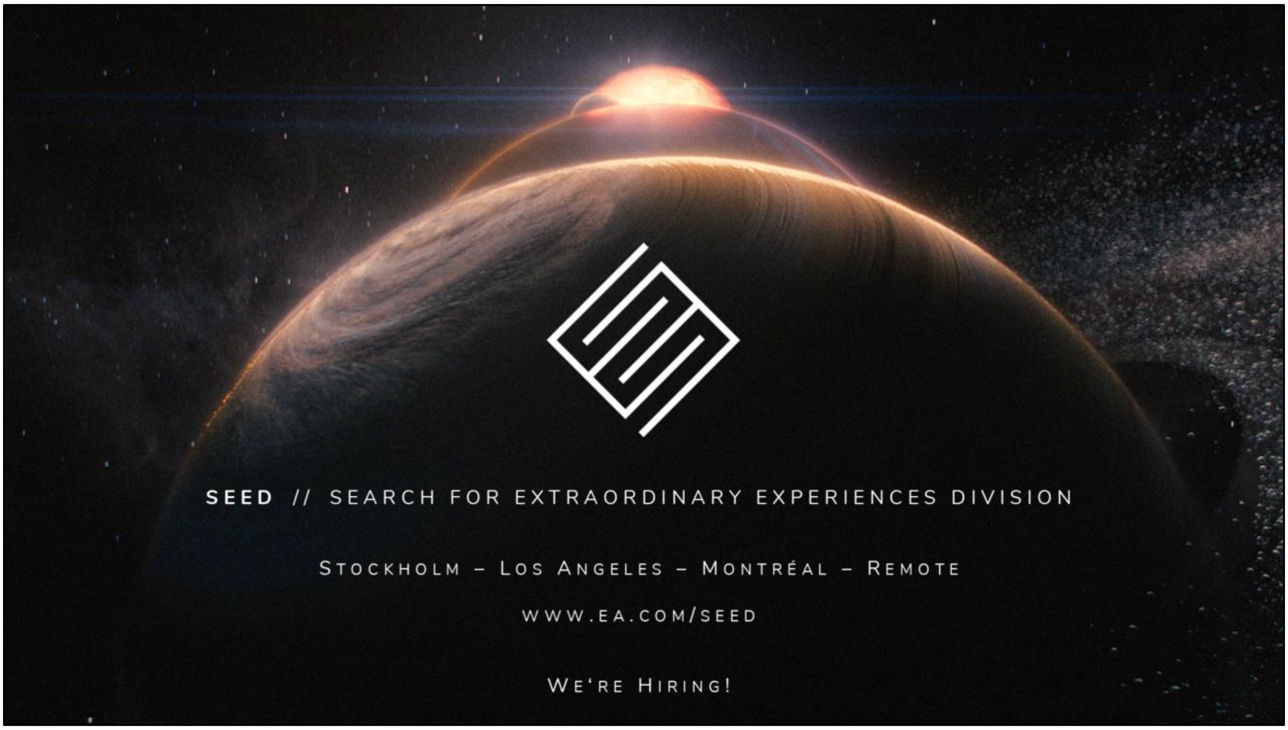
Colin Barré-Brisebois
SEED – Electronic Arts

Gathering Feedback: Open Problems in Real-Time Raytracing

For HPG 2018's keynote (co-located / two days before SIGGRAPH 2018) I'll be discussing some of the latest advances in game raytracing, but most ...

colinbarrebrisebois.com

Thanks to everyone who shared feedback!



SEED // SEARCH FOR EXTRAORDINARY EXPERIENCES DIVISION

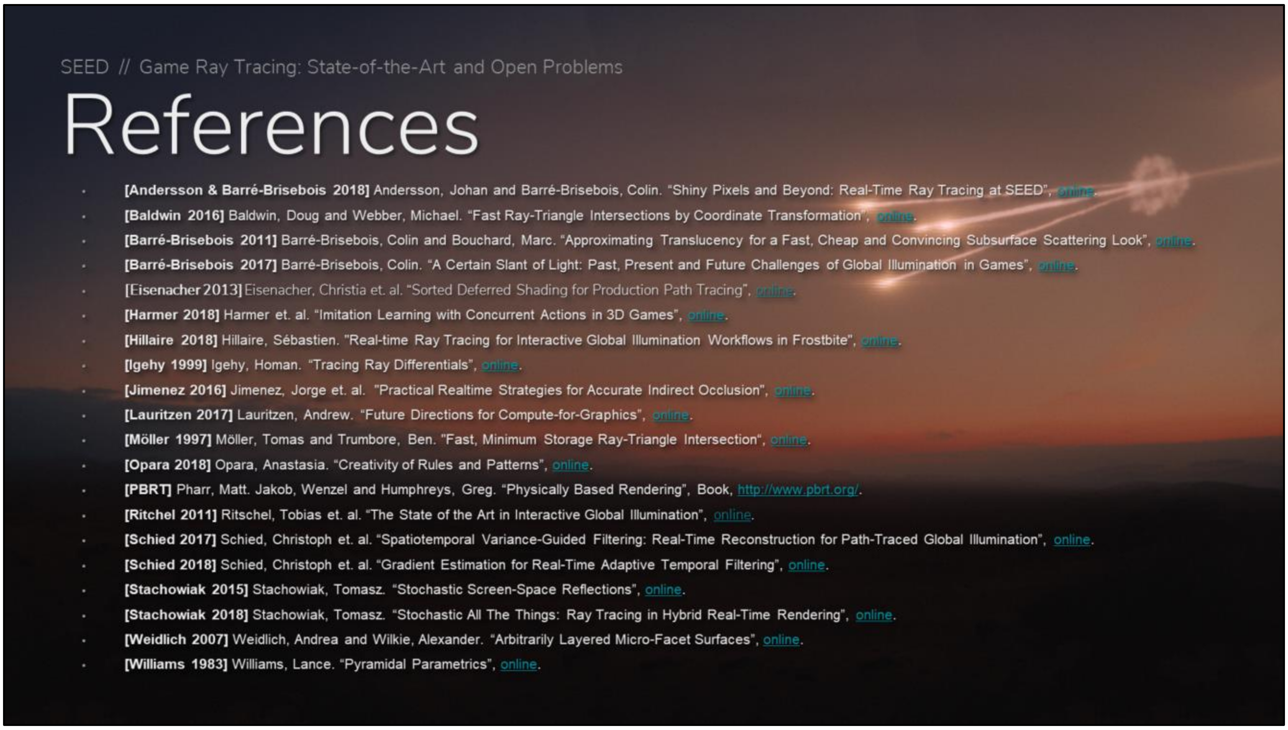
Stockholm – Los Angeles – Montréal – Remote

www.ea.com/seed

We‘re Hiring!

Questions?
Thoughts on RTRT?

I will now take questions, but mostly looking forward if people have any thoughts on real-time ray tracing challenges



References

- **[Andersson & Barré-Brisebois 2018]** Andersson, Johan and Barré-Brisebois, Colin. “Shiny Pixels and Beyond: Real-Time Ray Tracing at SEED”, online.
- **[Baldwin 2016]** Baldwin, Doug and Webber, Michael. “Fast Ray-Triangle Intersections by Coordinate Transformation”, online.
- **[Barré-Brisebois 2011]** Barré-Brisebois, Colin and Bouchard, Marc. “Approximating Translucency for a Fast, Cheap and Convincing Subsurface Scattering Look”, online.
- **[Barré-Brisebois 2017]** Barré-Brisebois, Colin. “A Certain Slant of Light: Past, Present and Future Challenges of Global Illumination in Games”, online.
- [Eisenacher 2013] Eisenacher, Christia et. al. “Sorted Deferred Shading for Production Path Tracing”, online.
- **[Harmer 2018]** Harmer et. al. “Imitation Learning with Concurrent Actions in 3D Games”, online.
- **[Hillaire 2018]** Hillaire, Sébastien. "Real-time Ray Tracing for Interactive Global Illumination Workflows in Frostbite", online.
- **[Igehy 1999]** Igehy, Homan. “Tracing Ray Differentials”, online.
- **[Jimenez 2016]** Jimenez, Jorge et. al. "Practical Realtime Strategies for Accurate Indirect Occlusion", online.
- **[Lauritzen 2017]** Lauritzen, Andrew. “Future Directions for Compute-for-Graphics”, online.
- **[Möller 1997]** Möller, Tomas and Trumbore, Ben. "Fast, Minimum Storage Ray-Triangle Intersection“, online.
- **[Opara 2018]** Opara, Anastasia. “Creativity of Rules and Patterns”, online.
- **[PBRT]** Pharr, Matt. Jakob, Wenzel and Humphreys, Greg. “Physically Based Rendering”, Book, http://www.pbrt.org/.
- **[Ritchel 2011]** Ritschel, Tobias et. al. “The State of the Art in Interactive Global Illumination”, online.
- **[Schied 2017]** Schied, Christoph et. al. “Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination”, online.
- **[Schied 2018]** Schied, Christoph et. al. “Gradient Estimation for Real-Time Adaptive Temporal Filtering”, online.
- **[Stachowiak 2015]** Stachowiak, Tomasz. “Stochastic Screen-Space Reflections”, online.
- **[Stachowiak 2018]** Stachowiak, Tomasz. “Stochastic All The Things: Ray Tracing in Hybrid Real-Time Rendering”, online.
- **[Weidlich 2007]** Weidlich, Andrea and Wilkie, Alexander. “Arbitrarily Layered Micro-Facet Surfaces”, online.
- **[Williams 1983]** Williams, Lance. “Pyramidal Parametrics”, online.