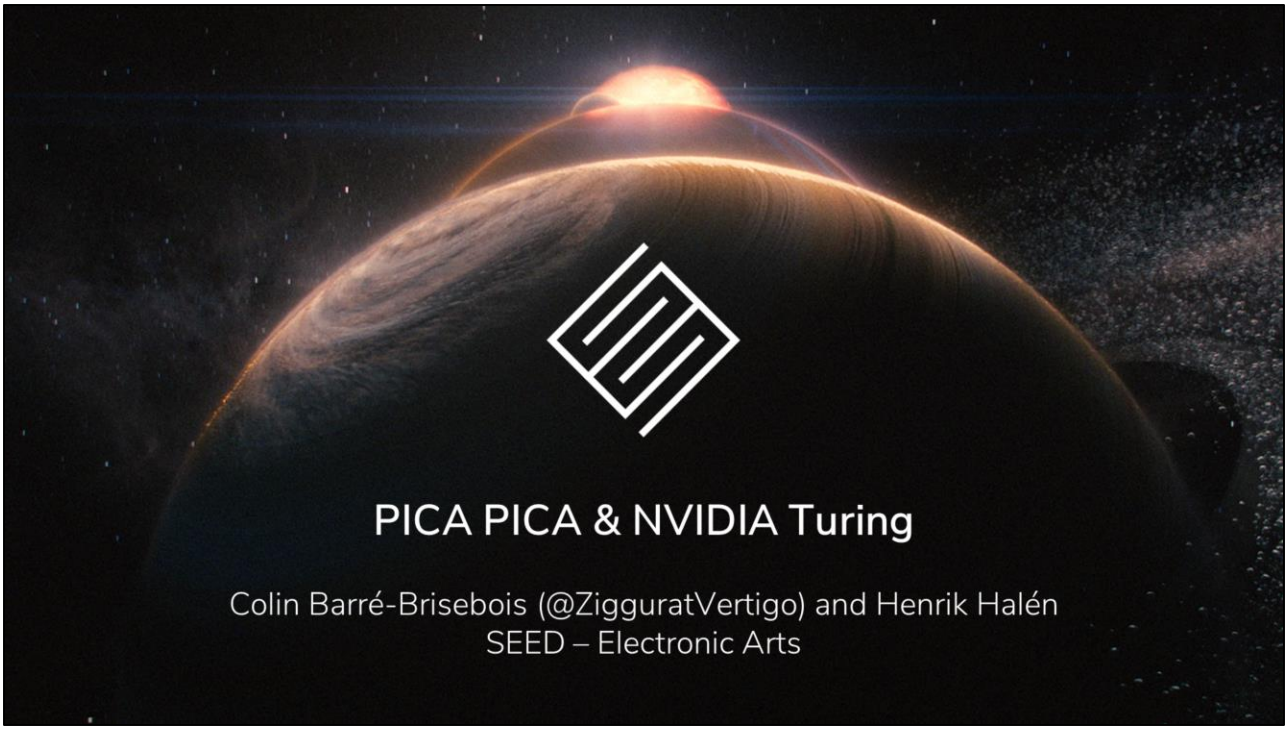
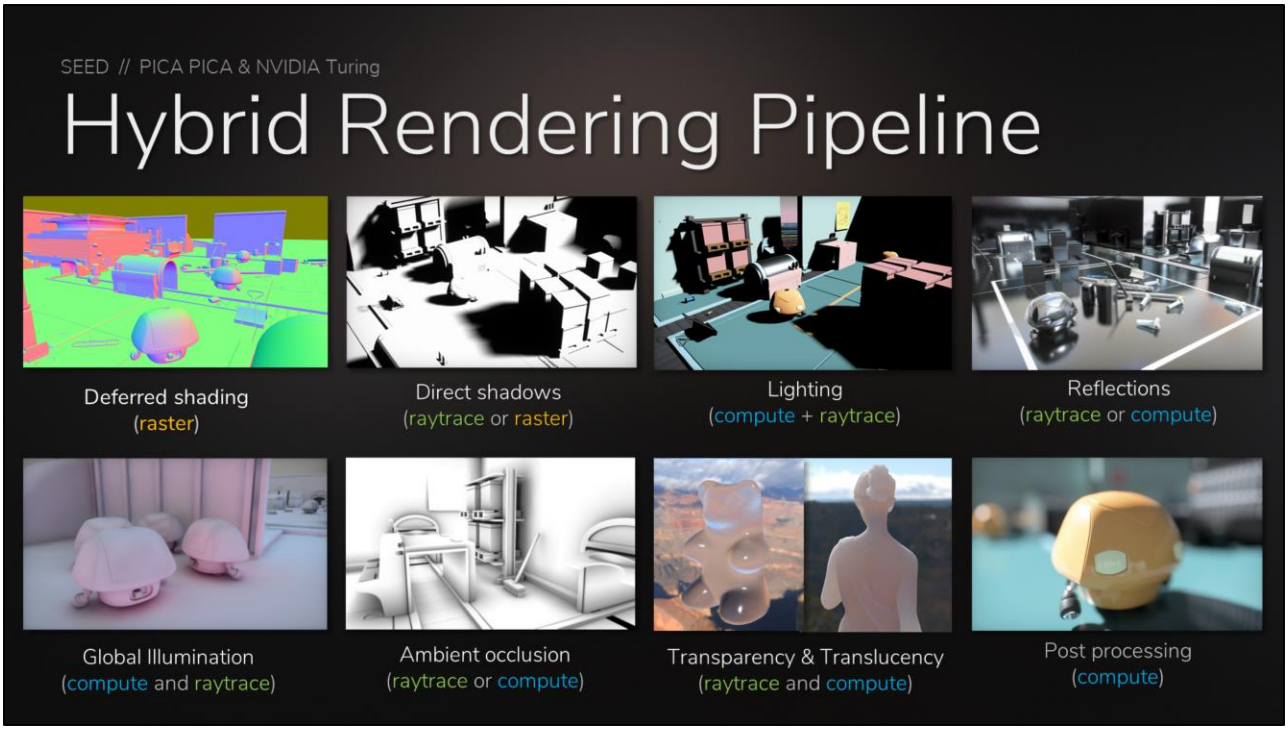


- Hi everyone, today I will give you an update on the raytracing work Henrik and I have we've been up to at SEED.
- For those of you who don't know, SEED is a technical and creative research division inside Electronic Arts
- One of our recent projects was an experiment with hybrid real-time rendering, 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!

- PICA PICA is a mini-game that we built for AI agents rather than humans.
- With 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've had the opportunity to be involved early on with DirectX Raytracing, thanks to our partners NVIDIA and Microsoft, to explore some of the possibilities behind this piece of 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 raytracing, 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.
- At GDC, we showed this on a single TitanV. We also support fork-and-join mGPU



- To build this demo we built a hybrid of classic rendering techniques with rasterization and compute, with some raytracing on top.
- We take advantage of the 3 pipelines, have a standard deferred renderer with compute-based lighting and a pretty standard post-processing stack.
- For example we can render shadows via raytracing or cascaded shadow maps.
- Reflections can be raytraced or screen-space raymarched. Same story for ambient occlusion.
- Global illumination and translucency requires raytracing.
- We’ve talked about the various techniques, so check out our previous talks
- This hybrid approach has allowed us to build visuals with almost path-traced quality in real-time on Volta

SEED // PICA PICA & NVIDIA Turing

Volta

- This scene @ 1080p on a TitanV
- Hybrid Rendering Mode

VOLTA (ms)			
SHADOWS			
1 spp		1.48	
2 spp		2.98	
4 spp		5.89	
8 spp		11.53	
16 spp		23.54	
AO			
	0.5m	2.0m	20m
1 spp	1.67	2.18	2.50
2 spp	3.41	4.48	5.08
4 spp	6.71	8.81	10.03
8 spp	13.27	17.44	19.85
16 spp	26.56	34.90	39.96
REFLECTIONS		2.97	
TRANS. & TRANSP.		0.47	
GI		1.70	

- This is the kind of performance we get on a single Volta
- The numbers on the right are for the scene displayed here to the left.
- This scene is rendered at 1080p with our hybrid raytracing mode
- For shadows and AO, this is sampled at 1 sample per pixel
- For reflections we do half a ray per pixel with temporal and spatial reconstruction
- I've added addition metrics with more sample per pixels just to give you an idea on how things scale
- I've also extended the default 0.5m radius for AO to 2 meters and 20 meters, just to see how things scale
- Obviously some of these workloads are not shippable, but you can see that they scale mostly linearly.

SEED // PICA PICA & NVIDIA Turing

Turing

Measured on **prototype hardware, early drivers** & **unmodified code**

VOLTA (ms)				TURING (ms)			x-faster
SHADOWS							
1 spp		1.48			0.44		**3.3X**
2 spp		2.98			0.77		**3.9X**
4 spp		5.89			1.31		**4.5X**
8 spp		11.53			2.33		**4.9X**
16 spp		23.54			4.65		**5.0X**
AO							
	0.5m	2.0m	20m	0.5m	2.0m	20m	
1 spp	1.67	2.18	2.50	0.54	0.62	0.62	**3.0 - 3.6X**
2 spp	3.41	4.48	5.08	0.88	1.01	1.01	**3.8 - 4.4X**
4 spp	6.71	8.81	10.03	1.48	1.64	1.64	**4.5 - 5.3X**
8 spp	13.27	17.44	19.85	2.55	3.02	3.02	**5.2 - 5.7X**
16 spp	26.56	34.90	39.96	4.90	5.82	5.82	**5.4 - 6.0X**
REFLECTIONS		2.97			1.45		**2.0X**
TRANS. & TRANSP.		0.47			0.25		**1.9X**
GI		1.70			0.35		**4.8X**

- If we map this a bit differently, with a performance scaling factor, some of the raytracing workloads are improved by up to 6X.
- In general it scales for 3x up to 6x compared to a volta at the same resolution
- This is really awesome!

SEED // PICA PICA & NVIDIA Turing

Turing

Higher quality, same budget:
GDC (1 spp) vs
SIGGRAPH (4-8 spp)

	VOLTA (ms)			TURING (ms)			x-faster
SHADOWS							
1 spp	1.48			0.44			**3.3X**
2 spp	2.98			0.77			**3.9X**
4 spp	5.89			1.31			**4.5X**
8 spp	11.53			2.33			**4.9X**
16 spp	23.54			4.65			**5.0X**
AO							
	0.5m	2.0m	20m	0.5m	2.0m	20m	
1 spp	1.67	2.18	2.50	0.54	0.62	0.62	**3.0 - 3.6X**
2 spp	3.41	4.48	5.08	0.88	1.01	1.01	**3.8 - 4.4X**
4 spp	6.71	8.81	10.03	1.48	1.64	1.64	**4.5 - 5.3X**
8 spp	13.27	17.44	19.85	2.55	3.02	3.02	**5.2 - 5.7X**
16 spp	26.56	34.90	39.96	4.90	5.82	5.82	**5.4 - 6.0X**
REFLECTIONS	2.97			1.45			**2.0X**
TRANS. & TRANSP.	0.47			0.25			**1.9X**
GI	1.70			0.35			**4.8X**

- Another way to look at this is the following: if we budget a certain amount of performance for a specific effect, basically we can have higher-quality visuals for the same budget
- For 1 sample per pixel on volta we can do 4-8 samples per pixel on turing
- This means faster reflections, translucency, AO, shadows and GI.
- As you saw from the video these effects are great when raytraced, and really add a lot

SEED // PICA PICA & NVIDIA Turing

Turing

Heavy workloads are still manageable

	VOLTA (ms)			TURING (ms)			x-faster
SHADOWS							
1 spp		1.48			0.44		**3.3X**
2 spp		2.98			0.77		**3.9X**
4 spp		5.89			1.31		**4.5X**
8 spp		11.53			2.33		**4.9X**
16 spp		23.54			4.65		**5.0X**
AO							
	0.5m	2.0m	20m	0.5m	2.0m	20m	
1 spp	1.67	2.18	2.50	0.54	0.62	0.62	**3.0 - 3.6X**
2 spp	3.41	4.48	5.08	0.88	1.01	1.01	**3.8 - 4.4X**
4 spp	6.71	8.81	10.03	1.48	1.64	1.64	**4.5 - 5.3X**
8 spp	13.27	17.44	19.85	2.55	3.02	3.02	**5.2 - 5.7X**
16 spp	26.56	34.90	39.96	4.90	5.82	5.82	**5.4 - 6.0X**
REFLECTIONS		2.97			1.45		**2.0X**
TRANS. & TRANSP.		0.47			0.25		**1.9X**
GI		1.70			0.35		**4.8X**

- Additionally if you look at the heavy workloads, with 8-16 samples per pixel, while expensive on Turing, performance is still manageable compared to similar settings on Volta.

SEED // PICA PICA & NVIDIA Turing

Transparency & Translucency

- Raytracing enables accurate light scattering
- Transparency
  - Order-independent (OIT)
  - Multiple index-of-refraction transitions
  - Variable roughness, refractions and absoption
- Translucency
  - Light scattering inside a medium
- We do this in texture-space
  - Handle view-dependent terms & dynamic changes to the environment

Texture-Space Glass and Translucency

- And so with all this new awesome power, we decided to revisit a technique that shines quite a lot with the power of raytracing: transparency and translucency
- Quickly, raytracing enables accurate light scattering for both transparency and subsurface translucency
- We've showed a technique that handles OIT, variable roughness, IOR transition as well as absorption
- For PICA PICA we did this in texture space, handing view-dependent and dynamic changes to the environment

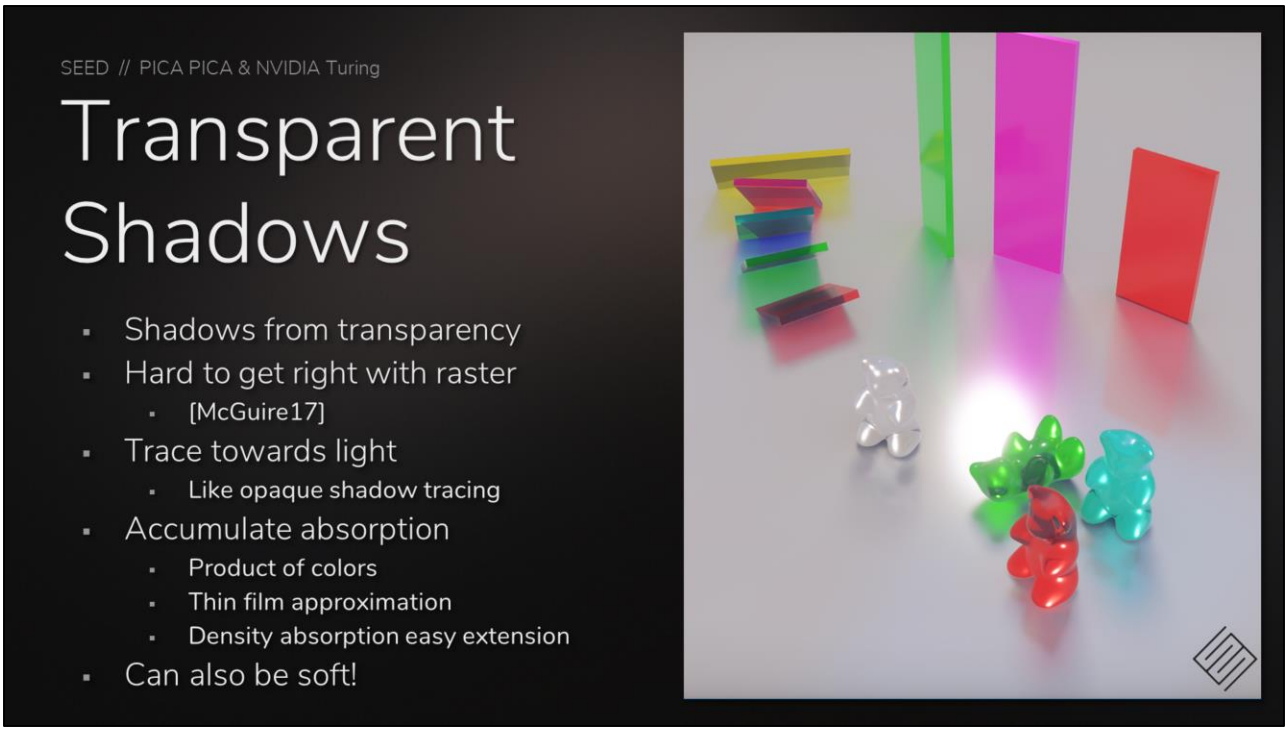
SEED // PICA PICA & NVIDIA Turing

Transparency & Translucency

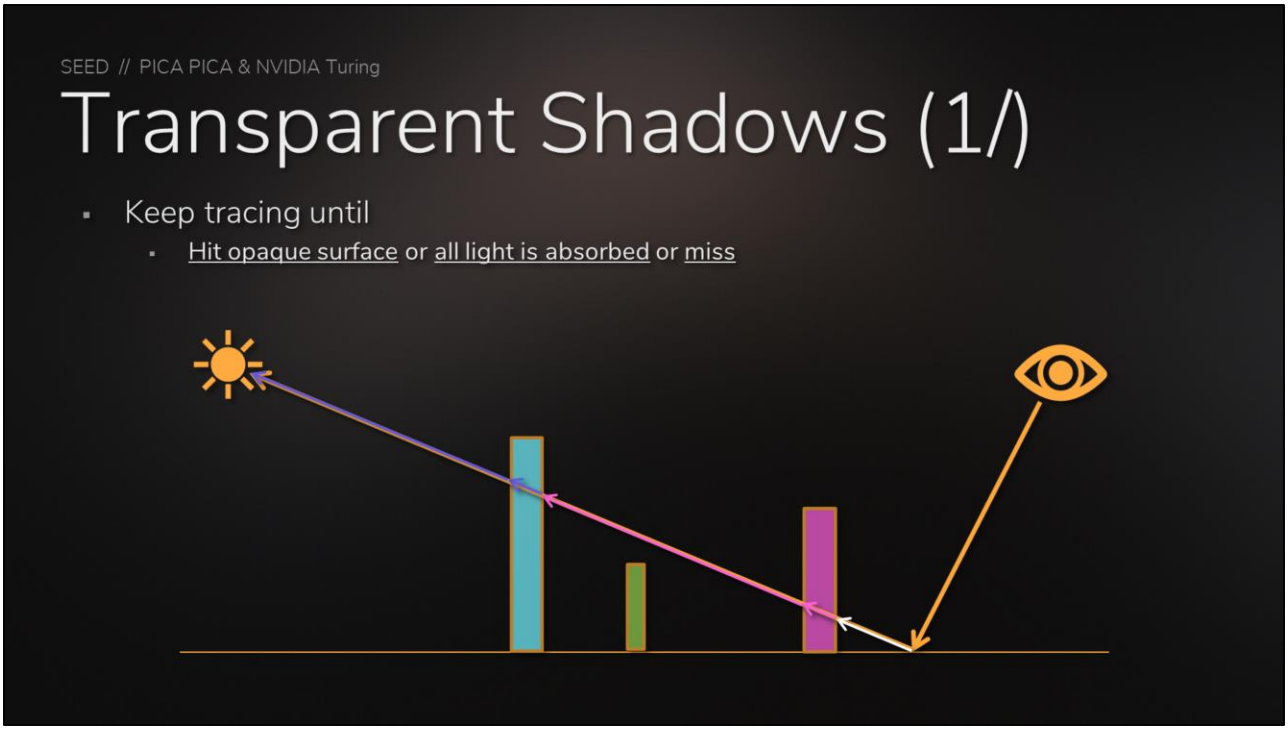
- Raytracing enables accurate light scattering
- Transparency
  - Order-independent (OIT)
  - Multiple index-of-refraction transitions
  - Variable roughness, refractions and absoption
- Translucency
  - Light scattering inside a medium
- We do this in texture-space
  - Handle view-dependent terms & dynamic changes to the environment

- And so with all this new awesome power, we decided to revisit a technique that shines quite a lot with the power of raytracing: transparency and translucency
- Quickly, raytracing enables accurate light scattering for both transparency and subsurface translucency
- We've showed a technique that handles OIT, variable roughness, IOR transition as well as absorption
- For PICA PICA we did this in texture space, handing view-dependent and dynamic changes to the environment

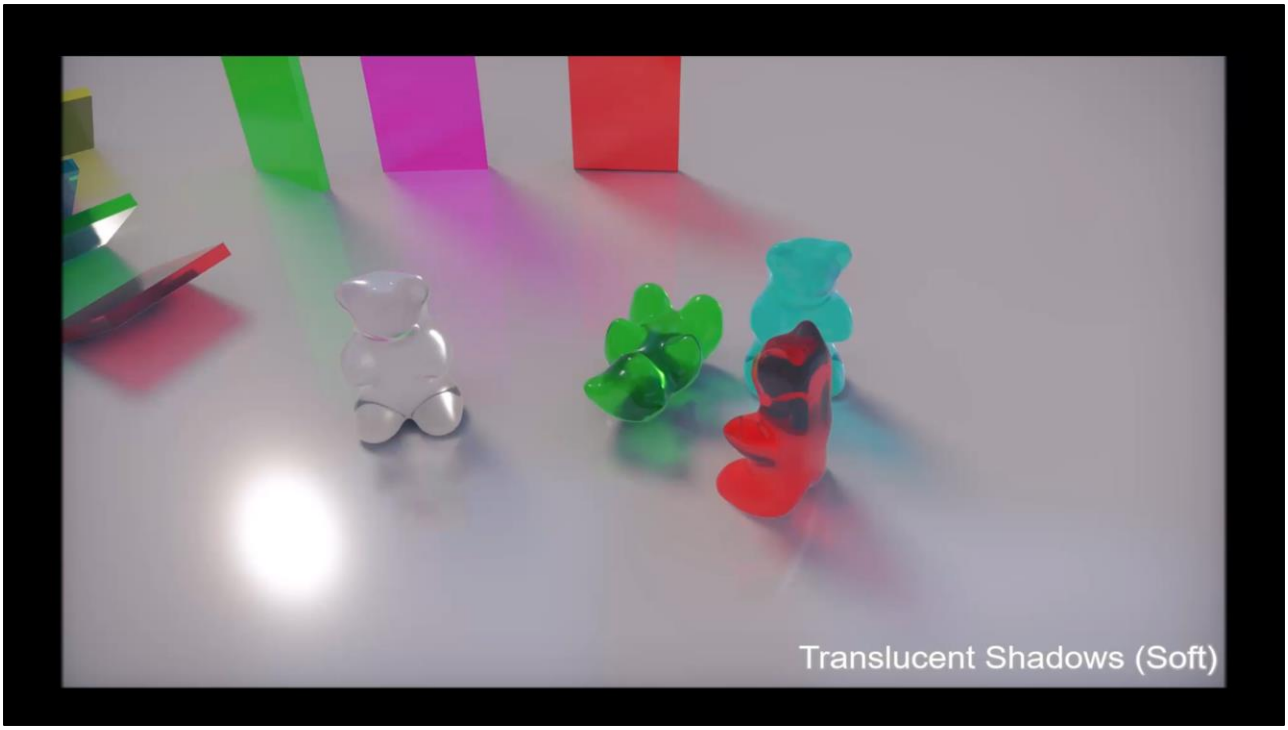
- But one thing that didn’t work so well was that we only supported opaque shadows
- If you look at the pen’s shadow, where the light goes through the medium and comes out, it looks quite wrong…



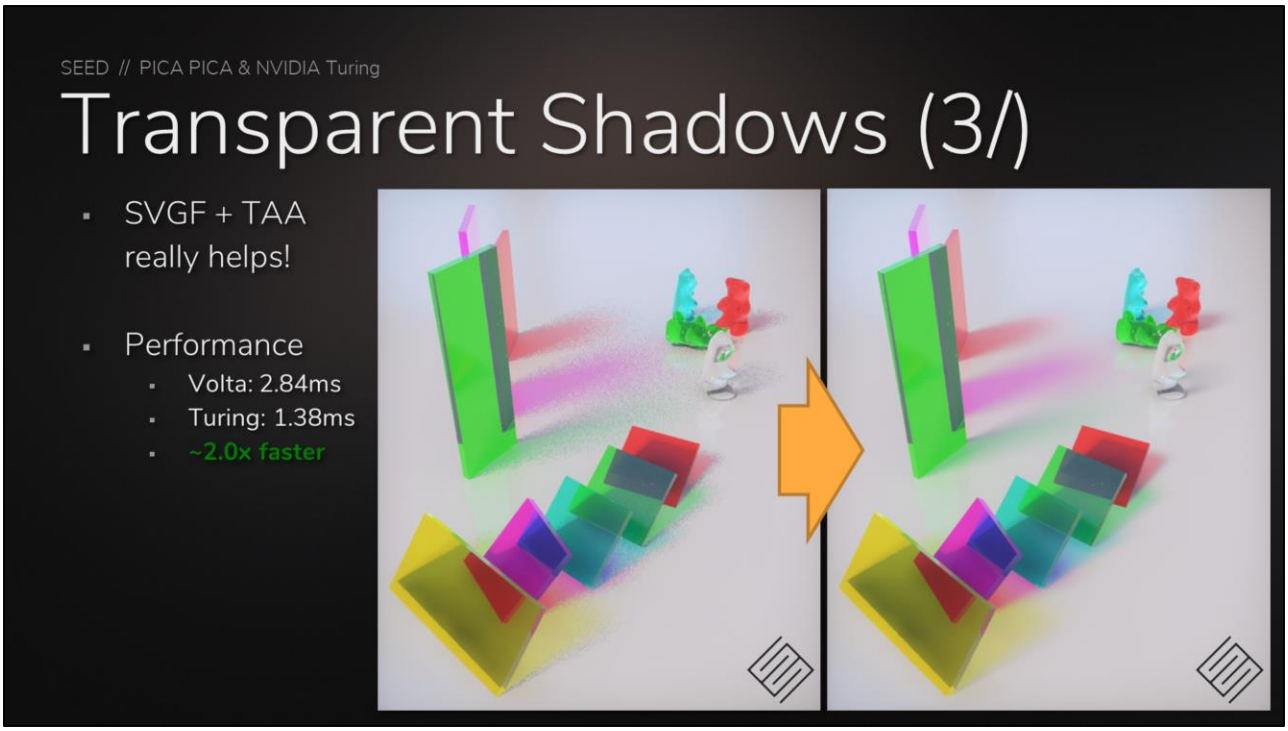
- And so we decided to implement transparent shadows
- Transparency is a hard problem in real-time graphics, especially if you are limited to rasterization
- With ray tracing some new alternatives are possible
- We achieve this by replacing the regular shadow tracing code with a recursive ray trace through transparent surfaces
- As the light travels through the medium we accumulate absorption
- Our current implementation treats this as a thin film approximation, where we assume all the color is on the surface
- Doing it properly is not trivial, but with some sensible limitations, it’s becomes possible in real-time
- Implementing distance-based absorption could also be an option that we have yet to try
- Just like our opaque shadows, transparent shadows can also be soft!



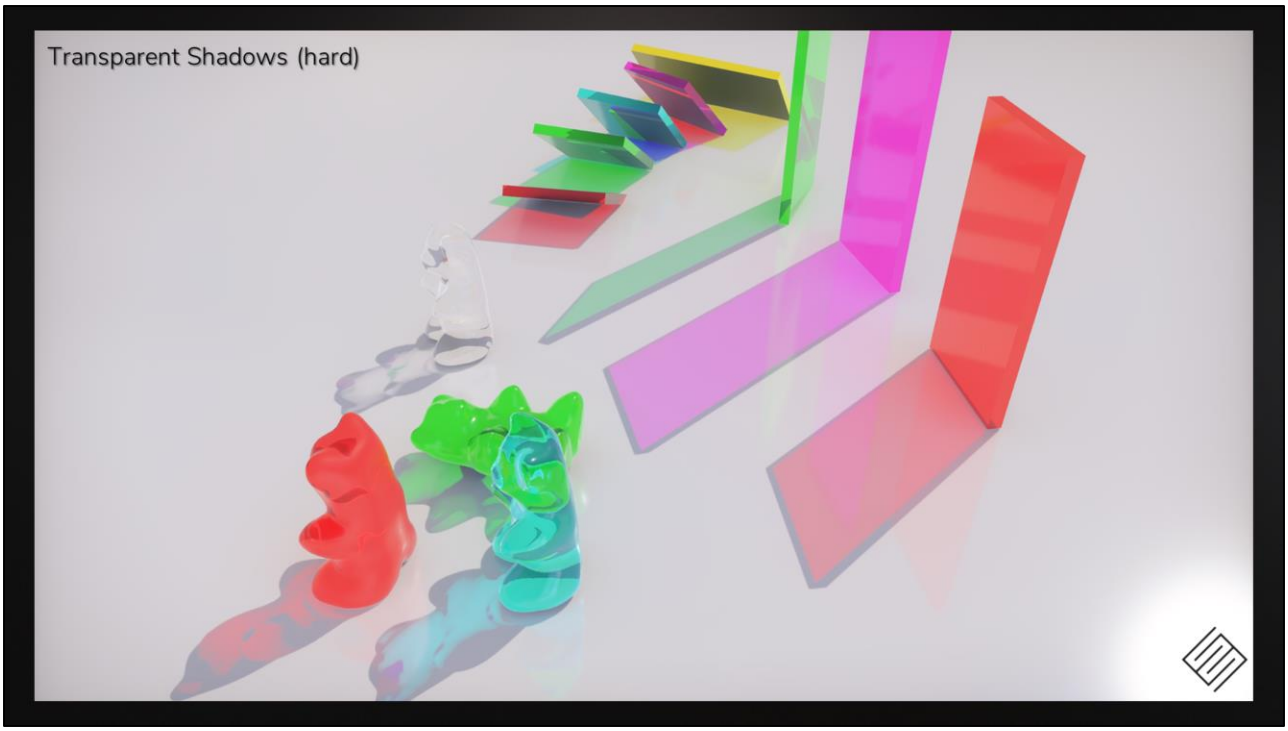
- This is a simple illustration of what happens
- For any surface that needs shadowing, we shoot a ray towards the light
- If we hit an opaque surface, or if we miss everything, we can stop
- If we hit a transparent surface however, we accumulate absorption based on the albedo of the object
- We also evaluate Fresnel. And then we keep tracing towards the light
- We keep doing this until 1. all light is absorbed, or 2. We miss in in the trace. 2. We hit an opaque surface



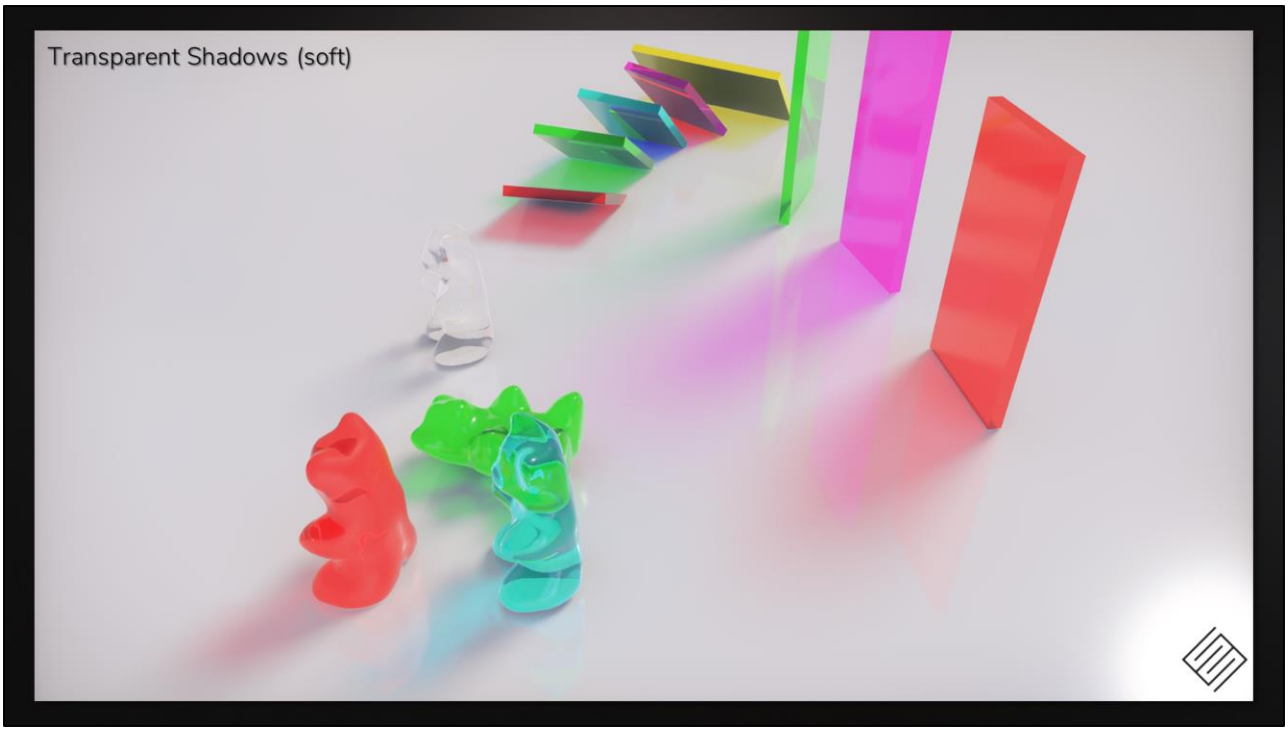
- This is not caustics, as we ignore caustic effects with out approach
- We do take Fresnel into account on the boundaries though
- It is also important to note that the regular Schlick approximation falls apart when the index of refraction on the incident side of the medium is higher than the far side
- We use Total-Internal-Reflection Fresnel, and filter the results with our tweaked SVGF that we mentioned in previous talks
- Additionally, in our implementation we only use this technique for direct shadows: any other pass that needs to sample light visibility use a simpler shadow pass that treats any surface as opaque to the light. A bit of a shortcut but it works for our case.



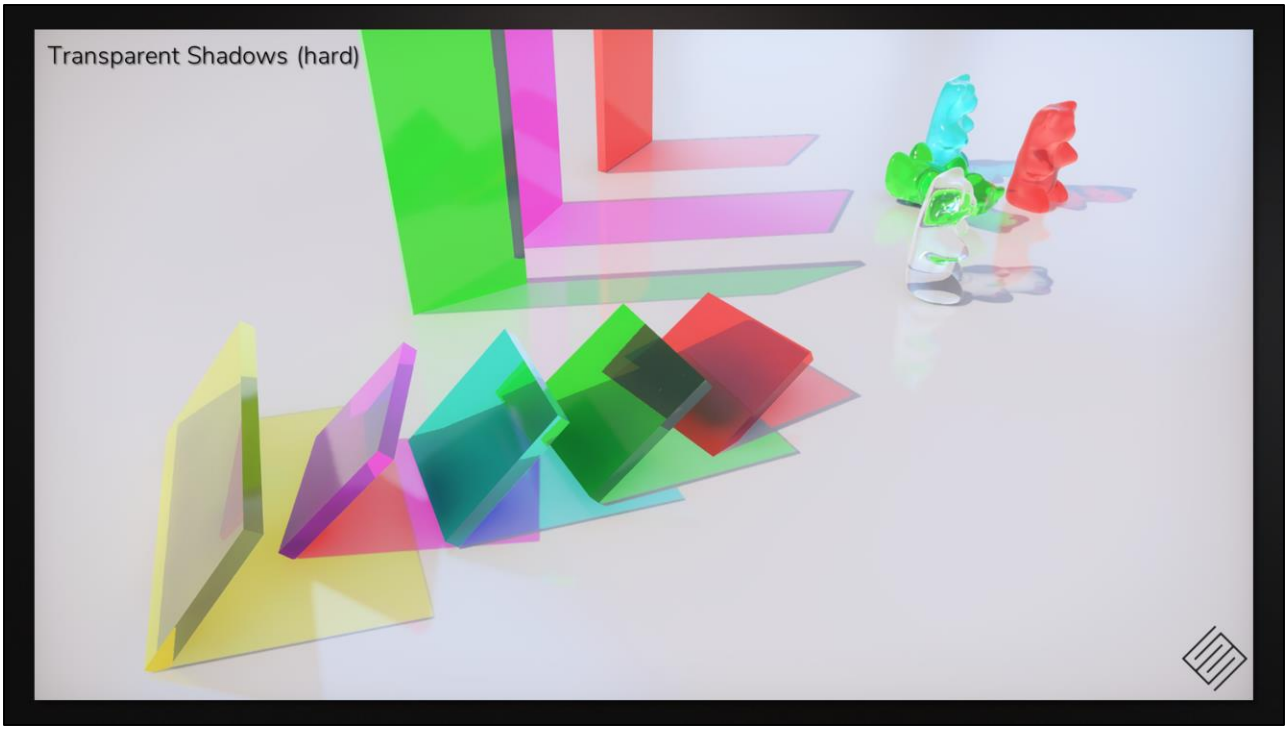
- We shoot one ray per pixel, per frame.
- This gets noisy, so like I said we use our modified SVGF and temporal anti-aliasing to filter
- And the performance is actually pretty good. So for this scene it’s about half the cost on Turing at 1080p compared to volta



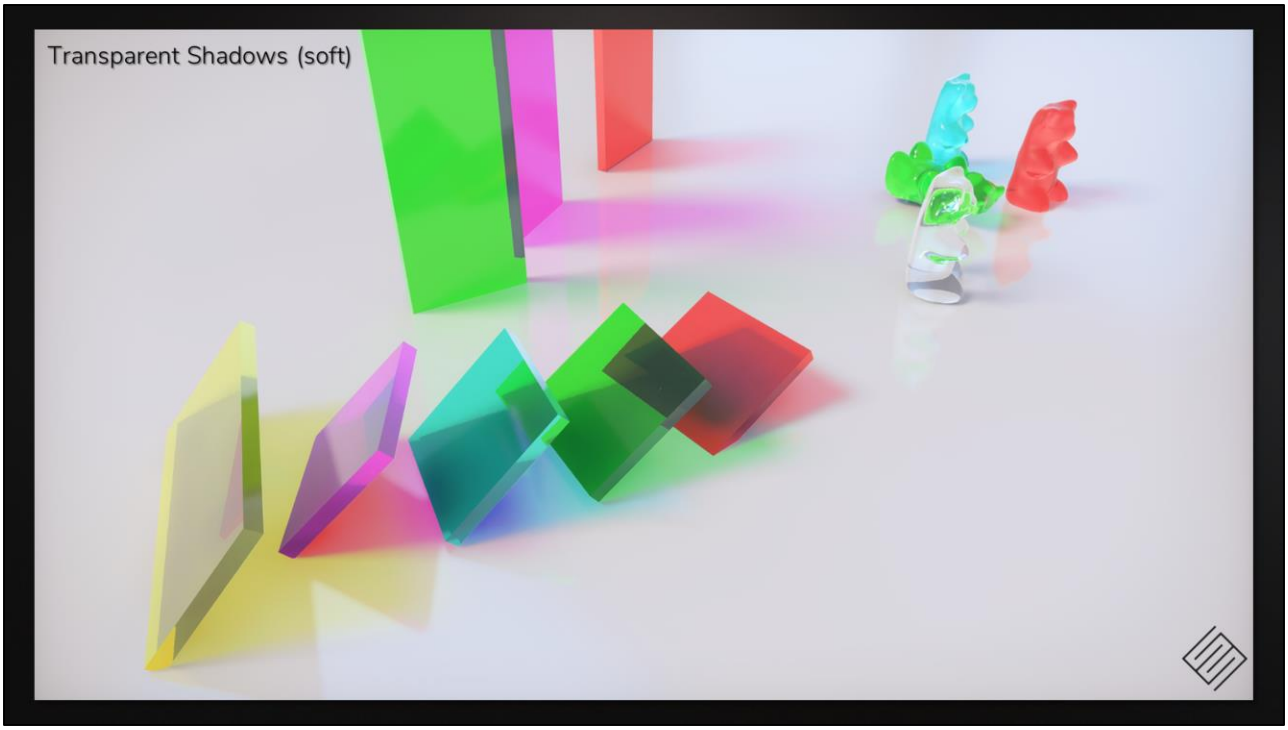
Here are some results, in hard-mode



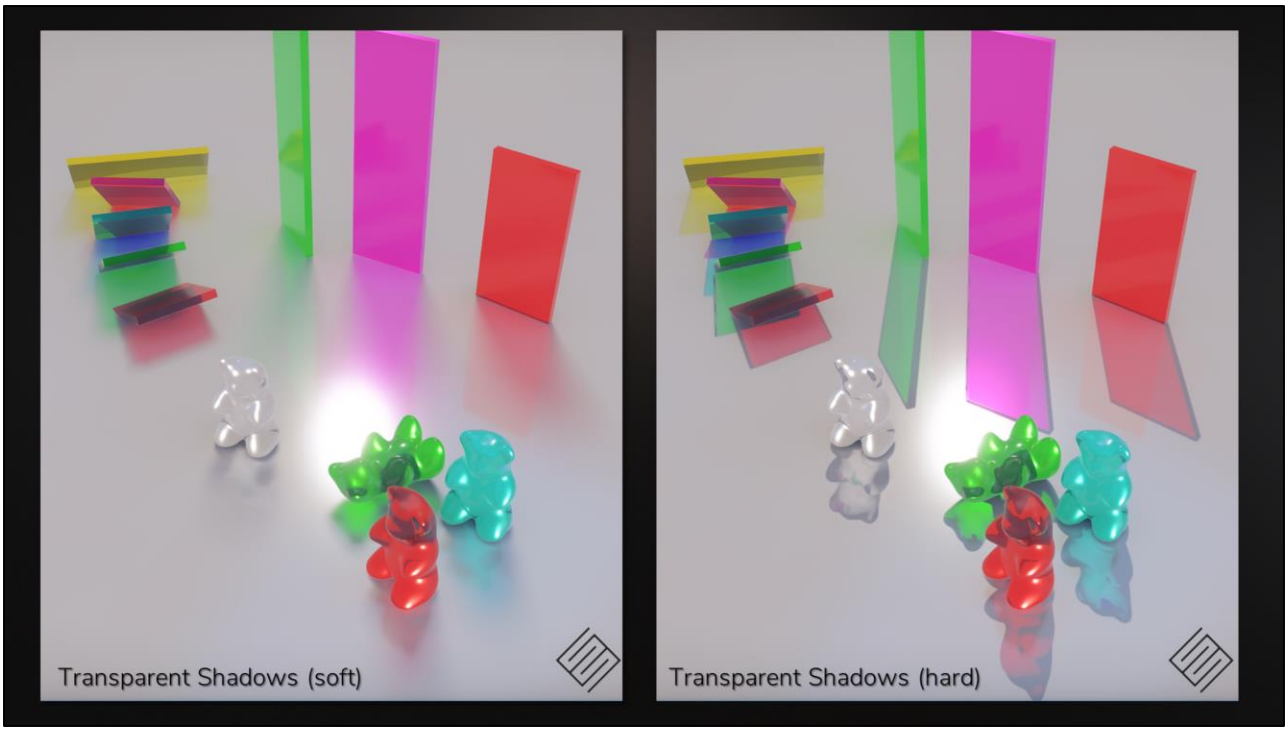
Now softly filtered



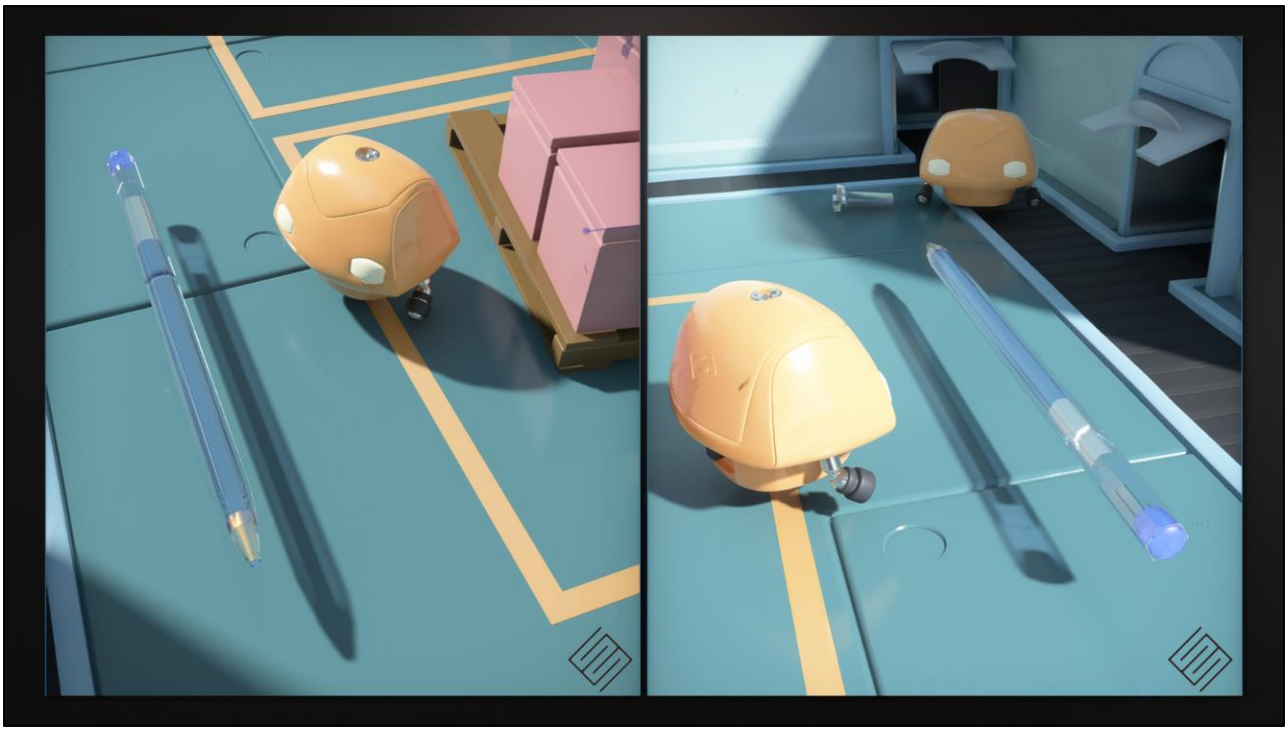
Again in hard-mode, from a different view.



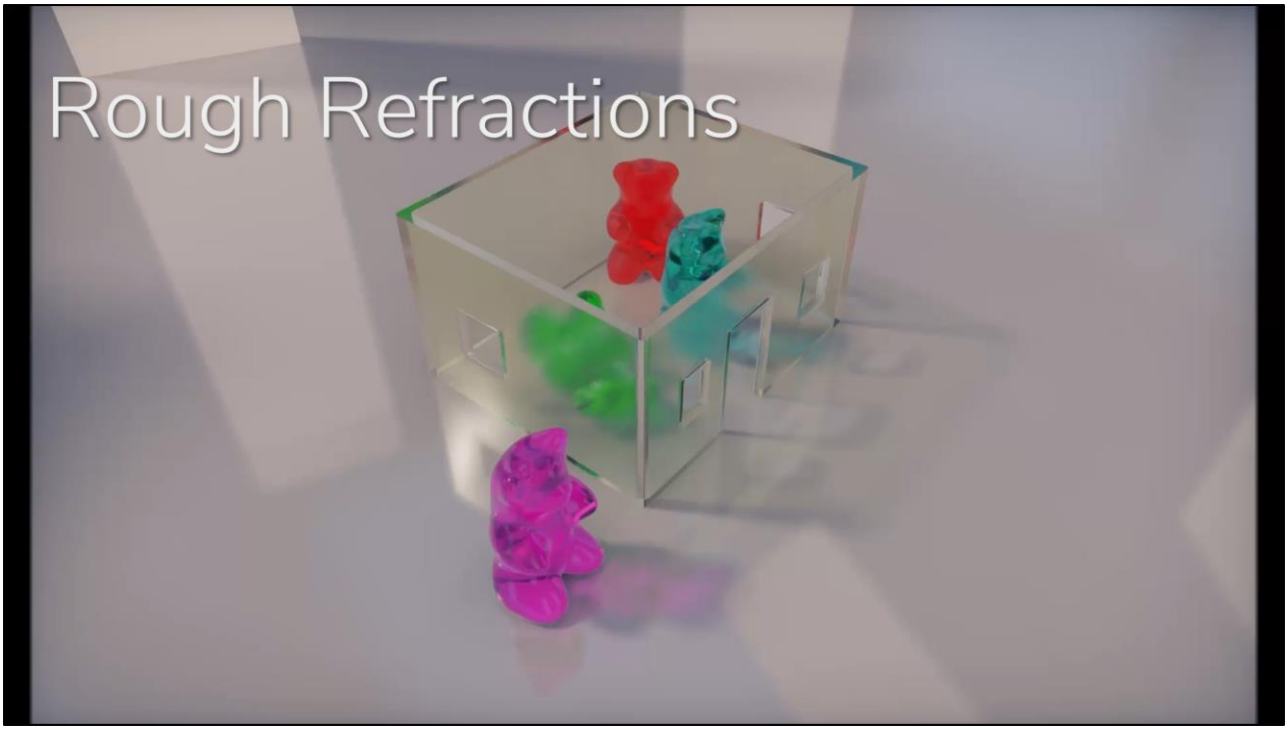
Soft-mode. Notice the colors blending



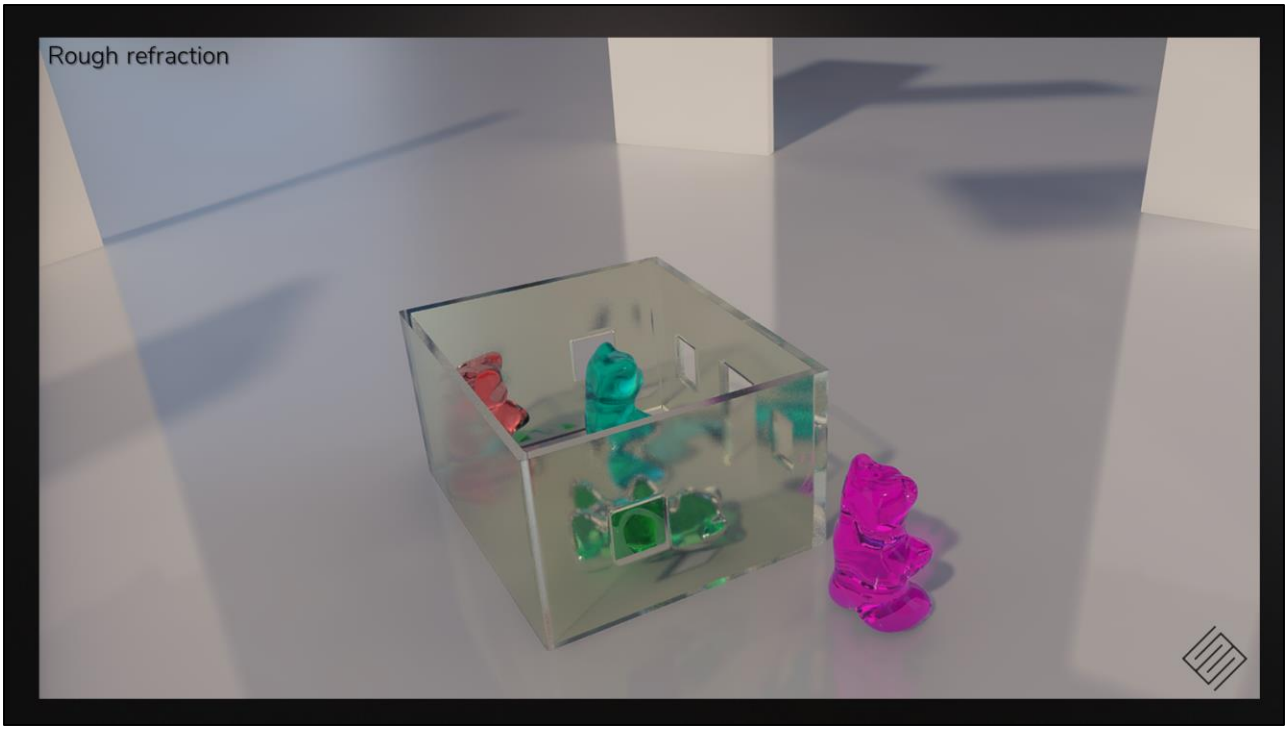
And now side-by side

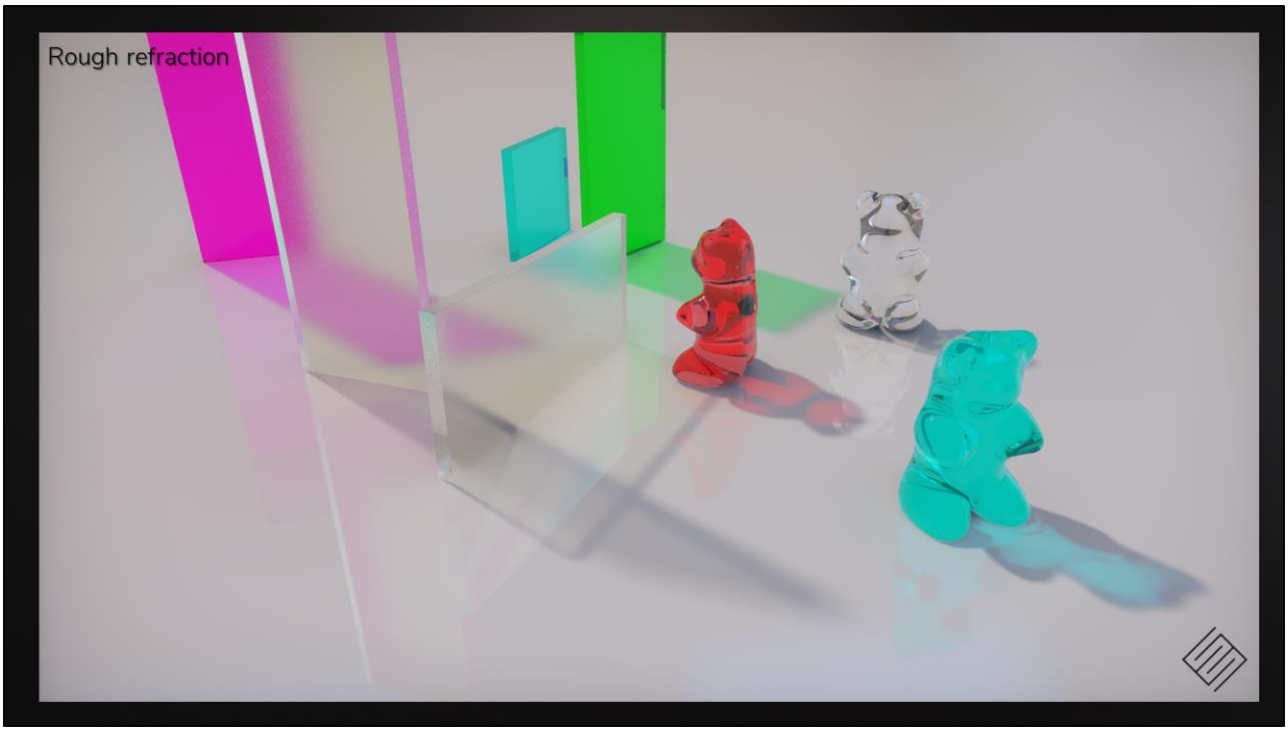


And most importantly we've now fixed out pen-shadow problem ☺ PHEW!

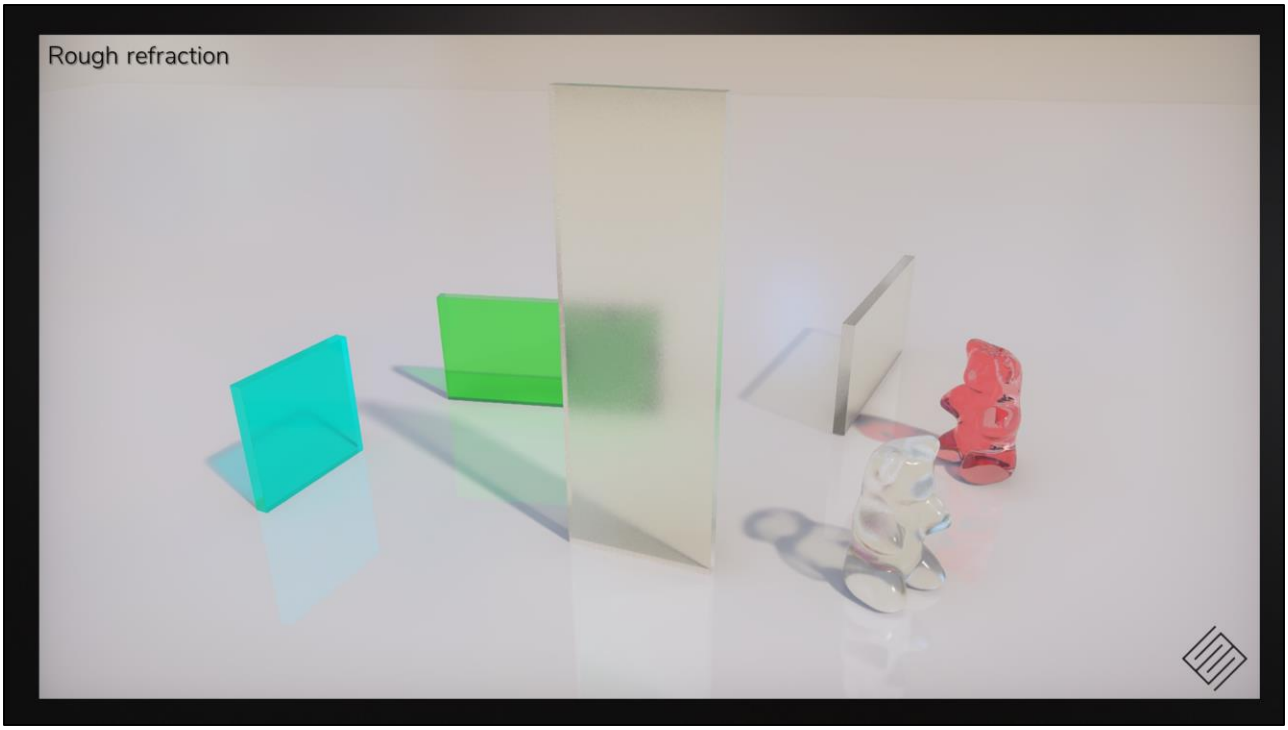


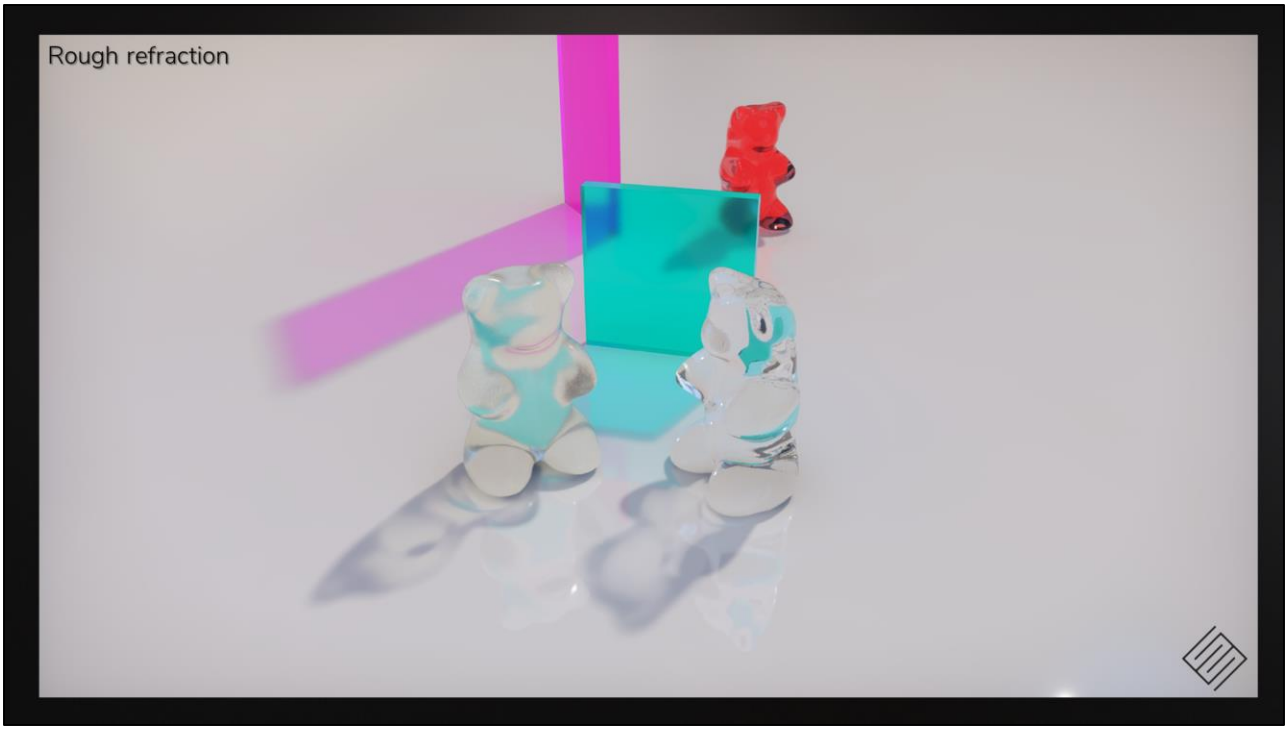
- As a bonus and quite last minute, we weren't super happy with the way we handle rough transparency
- In the previous approach we just linearly opened the cone based on roughness, kinda adhoc
- Microfacet models are pretty standard now for real-time applications, for opaque surfaces
- And so transparent surfaces, we needed a model that can handle refraction as well as reflection
- We implemented the microfacet distribution as described by Walter et al. to achieve this
- Walter handles the rough light interactions on air to object boundaries, but also provides physically accurate solutions for any surface-to-surface light transport
- We importance sample the GGX distribution, using this method, for both reflected and refracted rays on internal and external boundaries
- This is obviously more expensive, but still manageable and quite nice. We recently put this together and feel like we can optimize it more, but that will be for some other time
- Naïve Implementation @ 512x512 in texture space is 6.2ms on Volta, but 3.5ms on Turing


Rough refraction

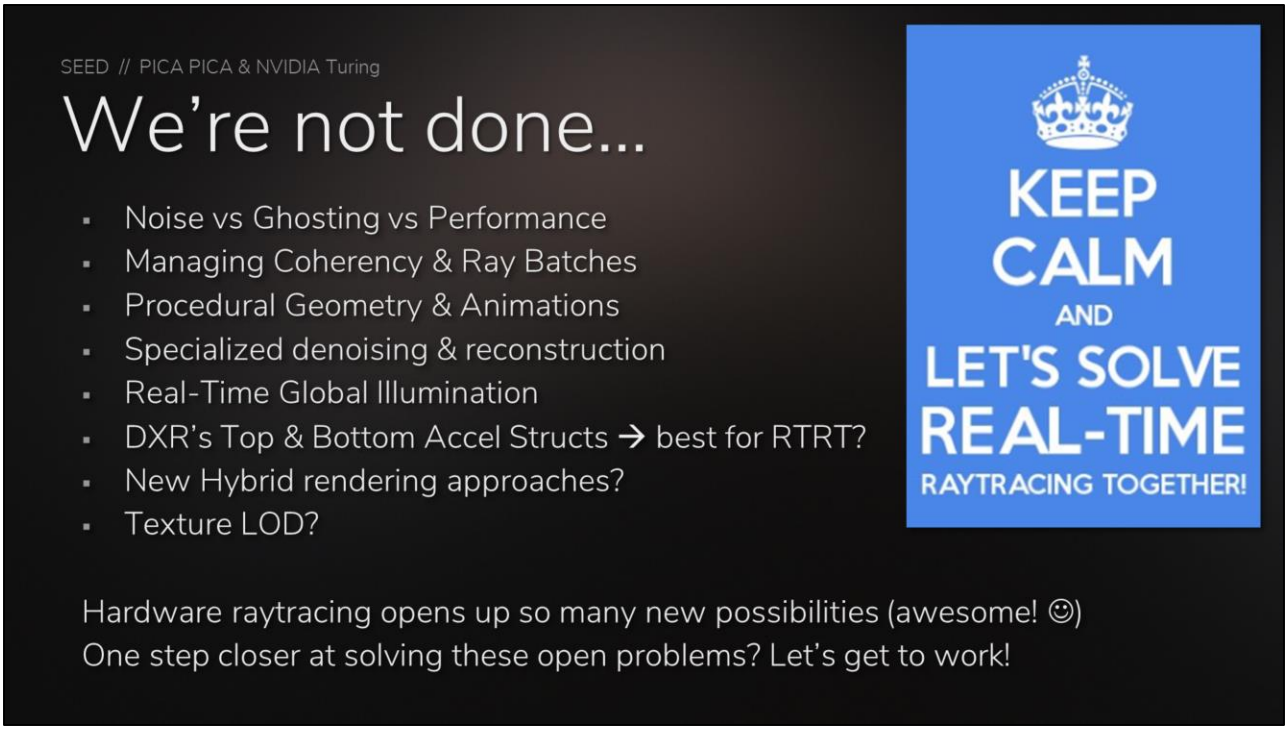


Here are the rough refractions


Rough refraction

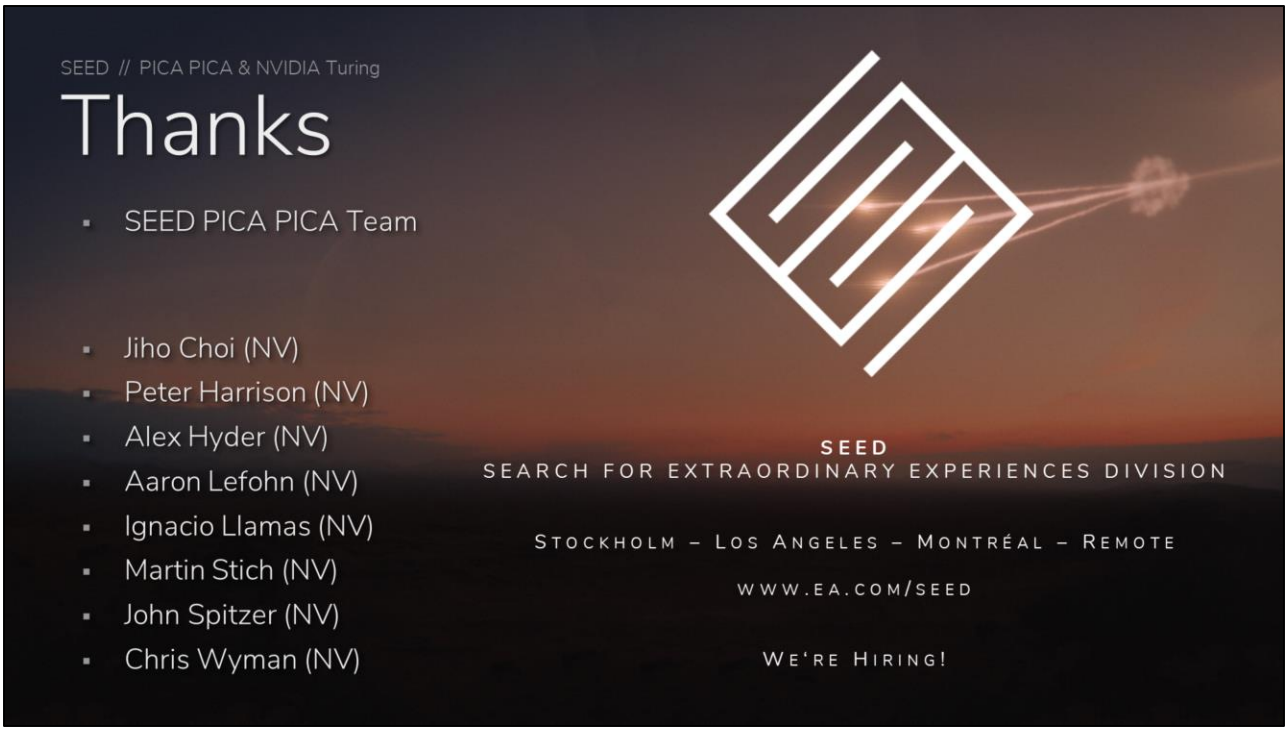


From another angle, notice the teal square and the glass gummy bears with different roughness



- And that's it for now
- This was just a glimpse on some of the stuff we've been up to lately
- Still have lots of open problem to solve with real-time raytracing, that we'll have to tackle together as an industry in the years to come!
- Hopefully some great collaborations between offline raytracing experts and the game industry!
- And now with hardware raytracing, so many new possibilities open up for real-time
- Let's get to work everyone! This is awesome!!

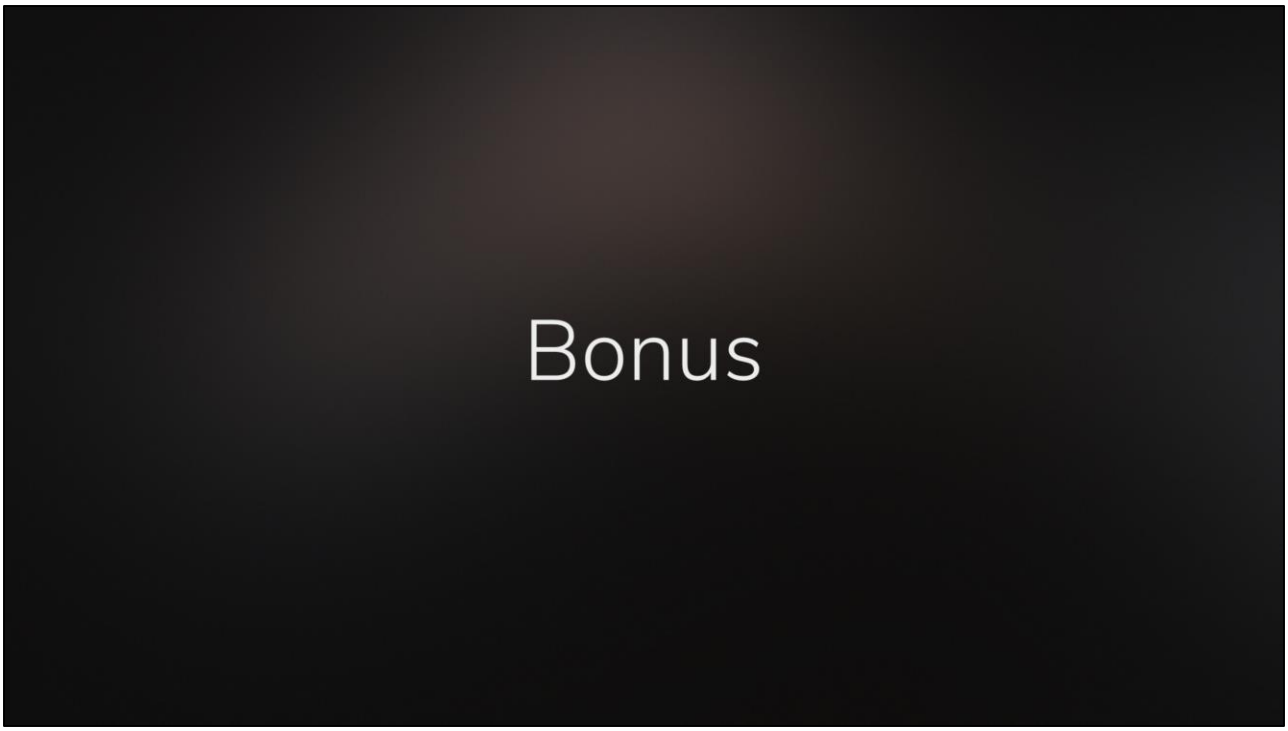
- Academia always asks for content from the games people and often doesn't get it.
- And so for SIGGRAPH we have decided to release all the assets from PICA PICA
- You can download them via Sketchfab and on our website, use them in your research for free, build your raytracing pipeline and compare with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

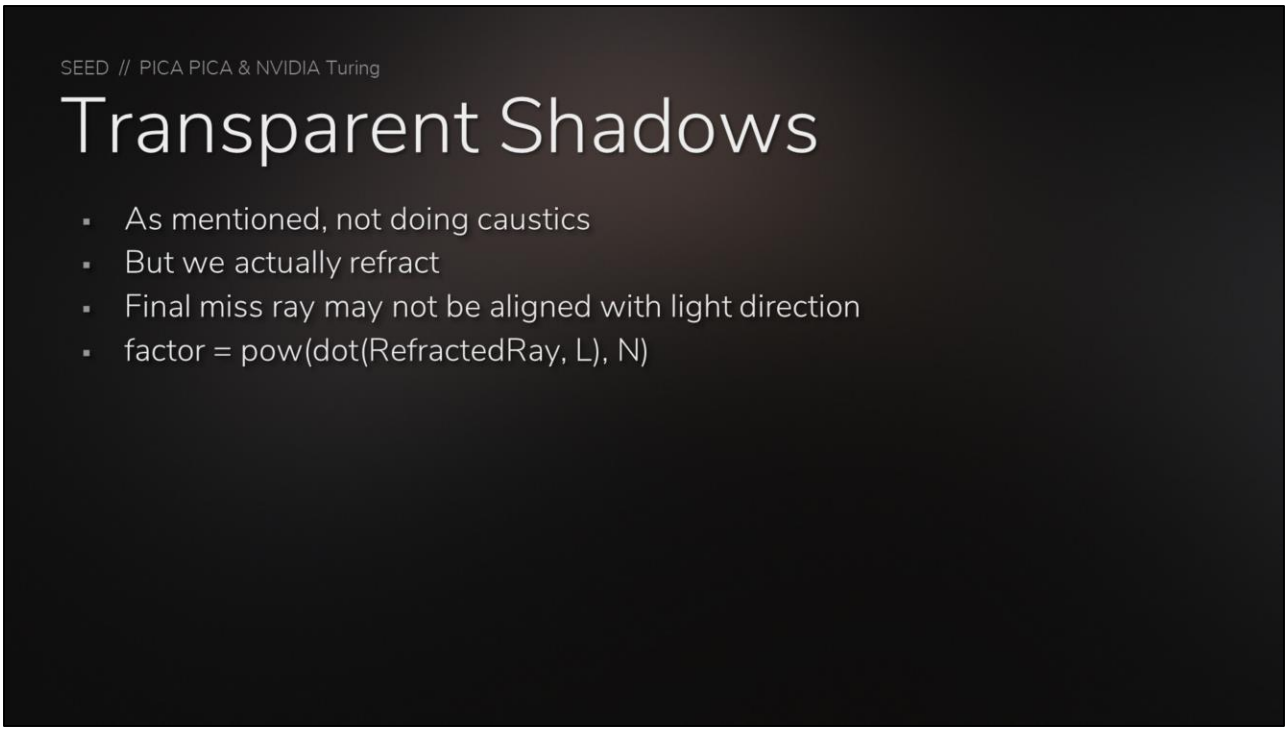


Check the bonus slides to see how we fake caustics

References

- **[Andersson & Barré-Brisebois 2018]** Andersson, Johan and Barré-Brisebois, Colin. "Shiny Pixels and Beyond: Real-Time Raytracing at SEED", online.
- **[Harmer 2018]** Harmer et. al. "Imitation Learning with Concurrent Actions in 3D Games", online.
- **[Lagarde 2013]** Lagarde, Sébastien. "Memo on Fresnel Equations", online.
- **[McGuire 2017]** McGuire, Morgan and Mara, Michael. "Phenomenological Transparency", online.
- **[Schied 2017]** Schied, Christoph et. al. "Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination", online.
- **[Stachowiak 2018]** Stachowiak, Tomasz. "Stochastic All The Things: Raytracing in Hybrid Real-Time Rendering", online.


Bonus



- As mentioned earlier we don’t do caustics for our shadows.
- However, we found a simple trick to make the shadows significantly more believable, a bit of a hack.
- Instead of just shooting rays straight towards the light for every refraction, we actually refract the ray.
- (And as mentioned before evaluate fresnel)
- This can end up with a ray that significantly deviates from the light direction.
- So if we miss, we dot the final ray direction with the light direction, with some power.
- This is completely not accurate, but provides a visually pleasing result.

SEED // PICA PICA & NVIDIA Turing

Transparent Shadows

- Refract and apply Fresnel
- factor = pow(dot(RefractedRay, L), N)