

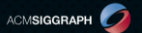
THE RENDERING OF
THE
CALLISTO
PROTOCOL

JORGE JIMENEZ
@iryoku1

MIGUEL PETERSEN
@miguel_oenp



Advances in Real-Time Rendering in Games course, SIGGRAPH 2023



Hello.



Jorge Jimenez

General Manager & Director of Creative Engineering

@iryoku1

My name is Jorge Jimenez. I am the Director and Head of Striking Distance Studios Spain.

In this talk we are going to present the rendering of The Callisto Protocol. A survival horror game released in 2022 using a modified version of Unreal Engine 4.

The game was designed as a linear AAA narrative experience, with emphasis in close quarter combat, characters and visual innovation.

This narrow but ambitious focus made for something special that we leveraged for the technical visuals of The Callisto Protocol.

I will play next a movie showcasing our game.

PHOTOREALISM

We worked in each piece of technology to elevate one of the main pillars of the game.

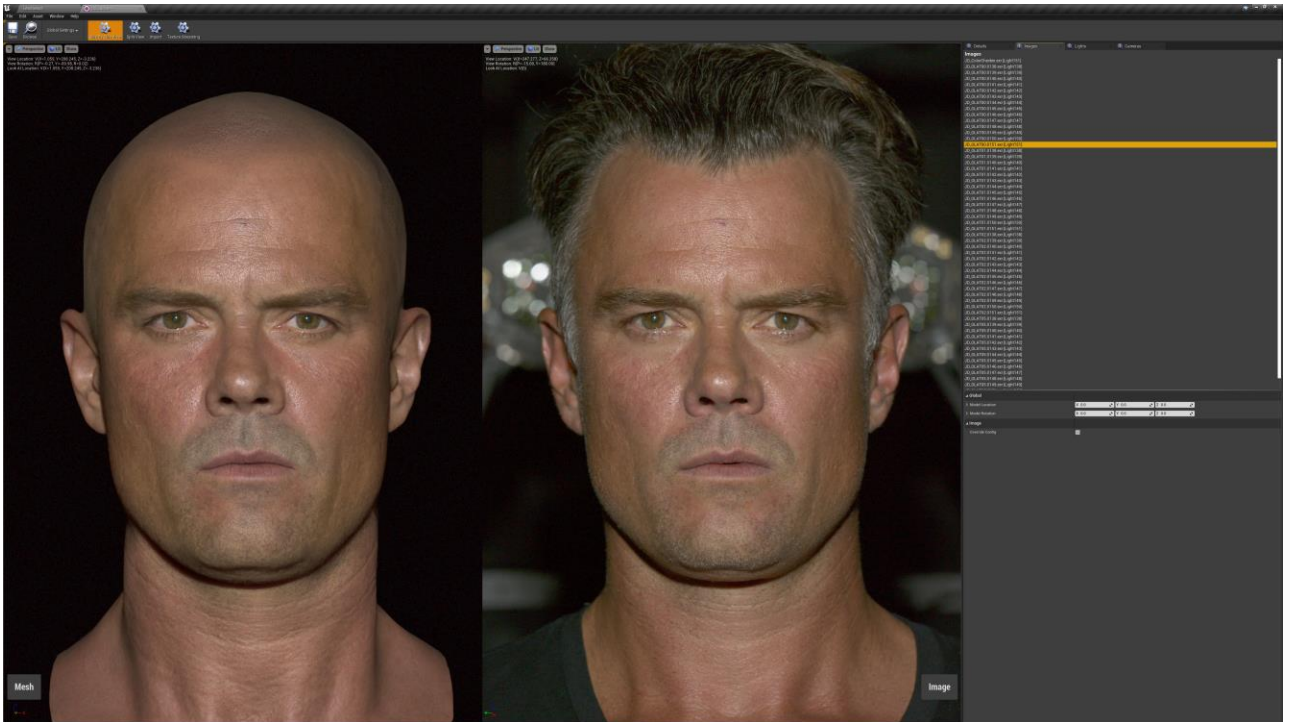
Photorealism.



Here we have a render of Jacob, our protagonist, compared with a reference photograph.

We pursued having the quality we see in this slide not only on cinematics, but during gameplay.

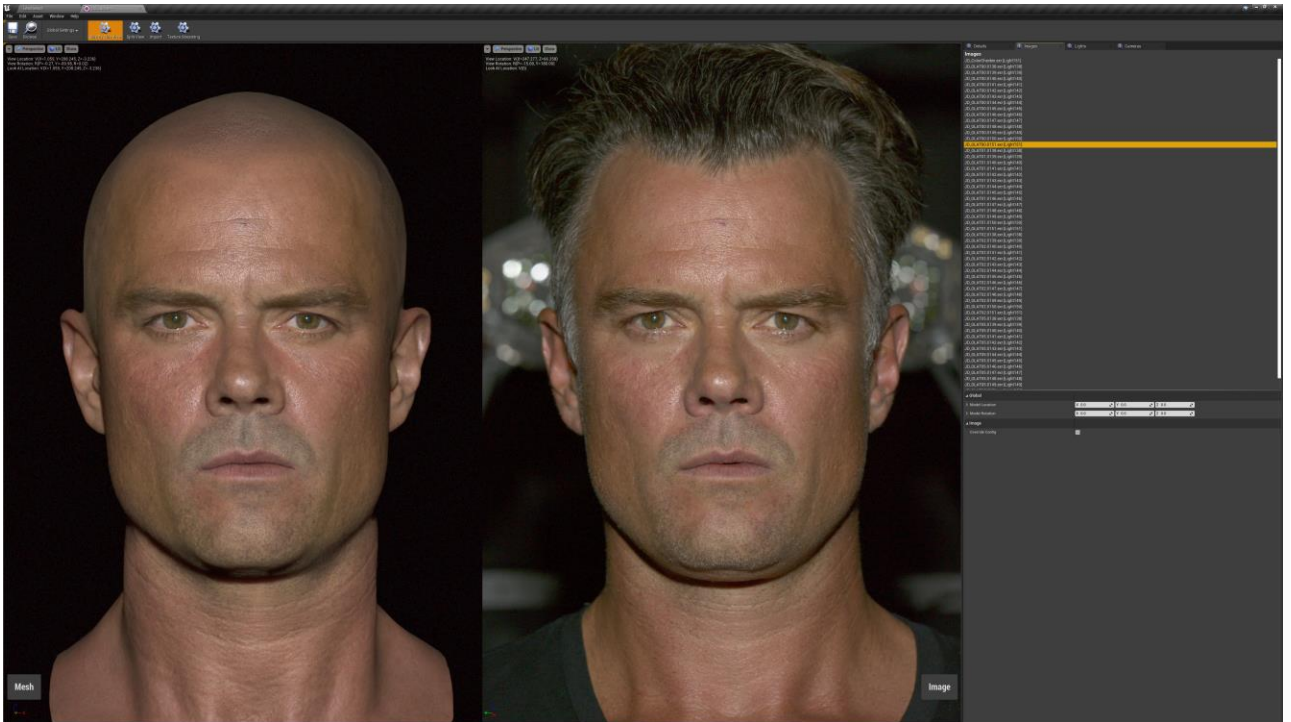
On any lighting. On any view condition.



On the previous render we composited the hair and background, as they were not part of the digital double process given the necessity of a different hairstyle.

We often do this process to visually isolate the areas we want to replicate.

In this slide we have the raw rendering results without the composited hair and background.



And this was the final version used in the game, where the jawline of the mesh was adjusted for stylization purposes.

This version will be used in the rest of this presentation.



This is another example, of our character Elias, with composited hair..



...and without.

Ground Truth.

Consistency.

Observation.

To achieve this quality, we based on three concepts that define our core foundation.

Ground truth.

Consistency.

And Observation.

OUR VISION

- **Ground Truth**

- Minimize shortcuts
- Pure runtime raytracing + Accurate cached lighting
- Long range light attenuation radius



Ground truth is the first goal we rally around.

It means for us to minimize the shortcuts taken in real time.

For that we heavily invested in raytracing, and specifically investigated how much we can do offline, before the game even runs, so that we can invest the runtime dollars where it matters the most.

Ground truth for us also means for lights to have large attenuation radiuses, allowing for natural and believable environments.

OUR VISION

- **Ground Truth**
 - Minimize shortcuts
 - Pure runtime raytracing + Accurate cached lighting
 - Long range light attenuation radius
- **Consistency**
 - Materials to respond correctly in all situations
 - Virtually all lights to have shadows (raytraced)
 - Virtually all surfaces to have accurate reflections (raytraced)



But what is ground truth if we cannot be consistent while executing the vision.

For materials to respond correctly in any view or light condition.

For all lights to have shadows, for them to be precise and using raytracing, and capable of representing the nuances of the lighting.

And finally for all surfaces to reflect light correctly, grounding the different elements into their surroundings, allowing for them to belong to a single unique and cohesive composition.

With this goal in mind, we relied in the most robust solution that exists today, which is raytracing. Which accurately predicts how light behaves in the real world.

OUR VISION

- **Ground Truth**
 - Minimize shortcuts
 - Pure runtime raytracing + Accurate cached lighting
 - Long range light attenuation radius
- **Consistency**
 - Materials to respond correctly in all situations
 - Virtually all lights to have shadows (raytraced)
 - Virtually all surfaces to have accurate reflections (raytraced)
- **Observation**
 - Train to discern subtleties
 - Photo reference
 - Digital Doubles



Which leads me to the last concept. Observation of our world. Because ground truth is not an absolute measure. It is a moving target, and changes as our understanding of the real world advances.

With photorealism in mind, our ultimate ground truth is the reality, rather than what our current state of the art models predict.

Observation is for us the art of learning how to discern the subtleties of lighting. The nuances of the materials and textures. The details that set apart what is synthetic, and what is real.

To understand them we invested in controlled capture environments. Which is where our digital doubles workflow was born. The process of taking an object of the real world, and turning it into a digital representation.

125+ milliseconds away.

[Note that the figure was updated since the official presentation to account for more up to date values]

Following ground truth, consistency and observation is not simple.

This philosophy set us 125 milliseconds away from our performance target.

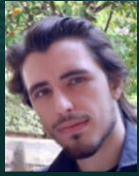
To put that number in perspective, it is what it takes to render 8 entire videogames at 60fps.

This, defined the start of our journey.

Rendering



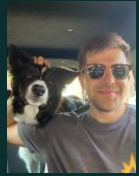
Jorge Jimenez
@iryoku1



Miguel Petersen
@miguel_oenp



Miguel Rodriguez
@MiguelRodRic

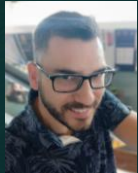


Jose Naranjo
@_aborres



Fran Nuñez
@loennirdev

Core



Ruben Segura
rubenseguramayor



Jon Diego
@CrazyMoai



Jesus Serrano
jesusserranogarcia



Isaac Lascasas
@double_coder



Edu Martin
@edumarting

Tech Art



Martin Contel
@martincontel



Pablo J de Andres
@pablojdeandres

A journey that could only succeed thanks to this team.

Three years ago, we opened a studio in Zaragoza, Spain, with the main goal of pursuing visuals innovation in videogames.

I will cover in this presentation the direction, the philosophy and core concepts we used to pursue photorealism, and Miguel Petersen will follow up with the challenges we found and overcame as the project advanced.



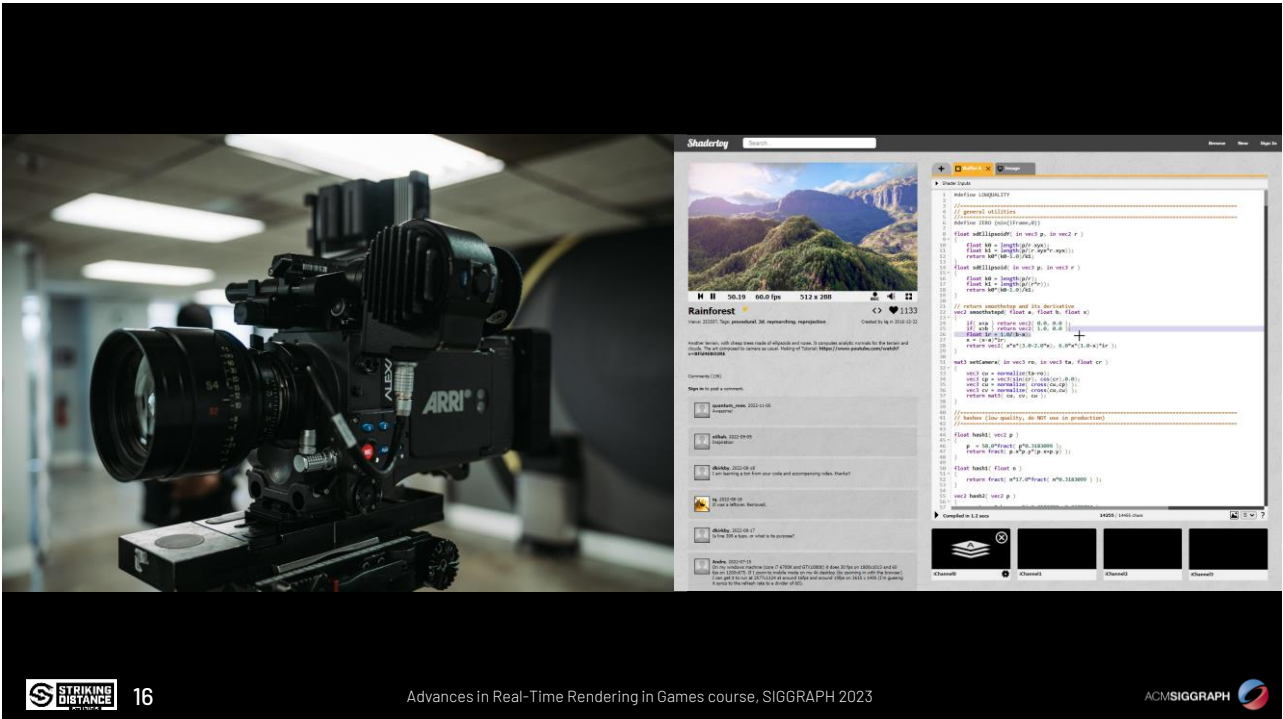
Building a traditional clock is a unique challenge.

You need to blend the engineering behind the mechanics, with the artistic choices of the aesthetics.

You need to understand both art and engineering to really master the design.

This is core to how we approached characters in the Callisto Protocol, and in general, the technical visuals in our game.

With art and technology working as one.



I found over the years that a unique talent one might have, is to be able to understand the varied processes behind digital humans creation, such as:

Capturing data, cleaning, processing and altering textures and models,

or working with pixel shaders.

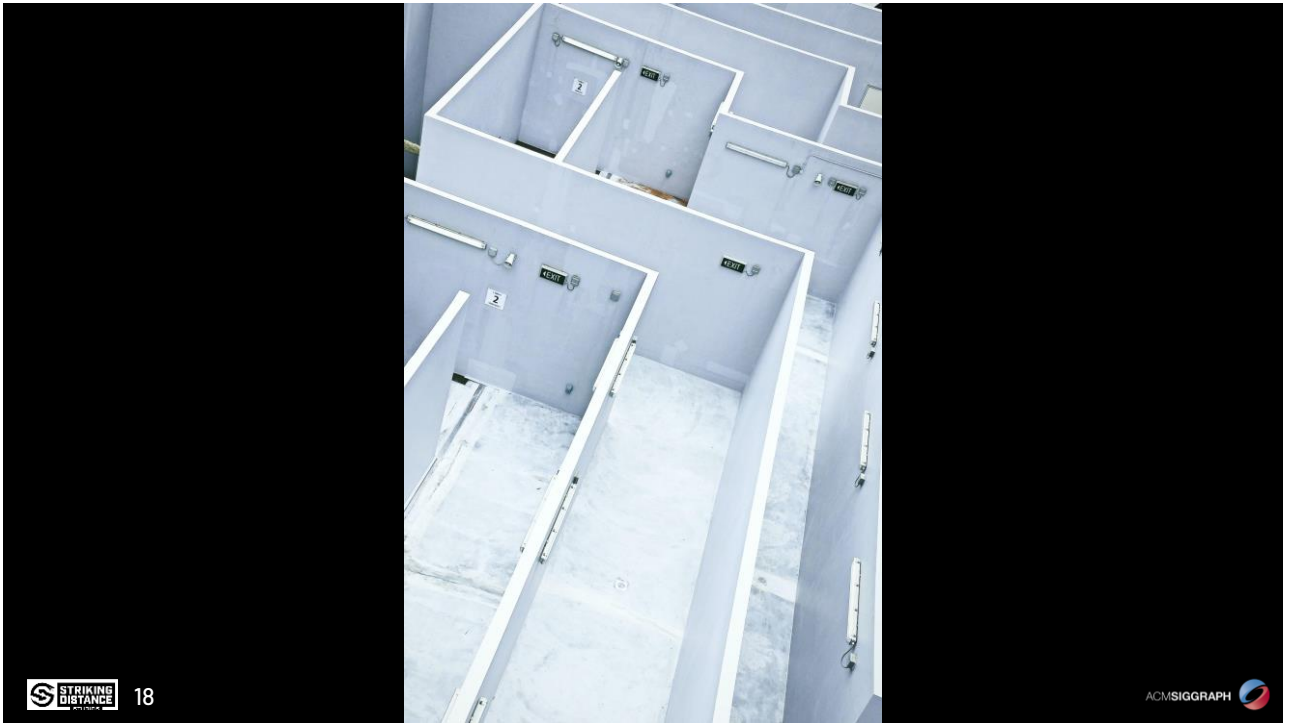
I call this, controlling from captured texel to rendered pixel, and was key to achieve photorealistic looking characters in our game.

For engineering to cross the boundary of art, and for artists to grow curiosity on the technologies they work with. This is possibly one of the first challenges we find when pursuing photorealism.



But it is not the only one. People often look for the key feature, the key solution to unlock photorealism.

As in these doors, thinking one of them is hiding the treasure that will made us overcome the uncanny valley.



Reality is, that it is not one door but many.

It is a maze where we have to make many turns, and where each turn must be made correctly.

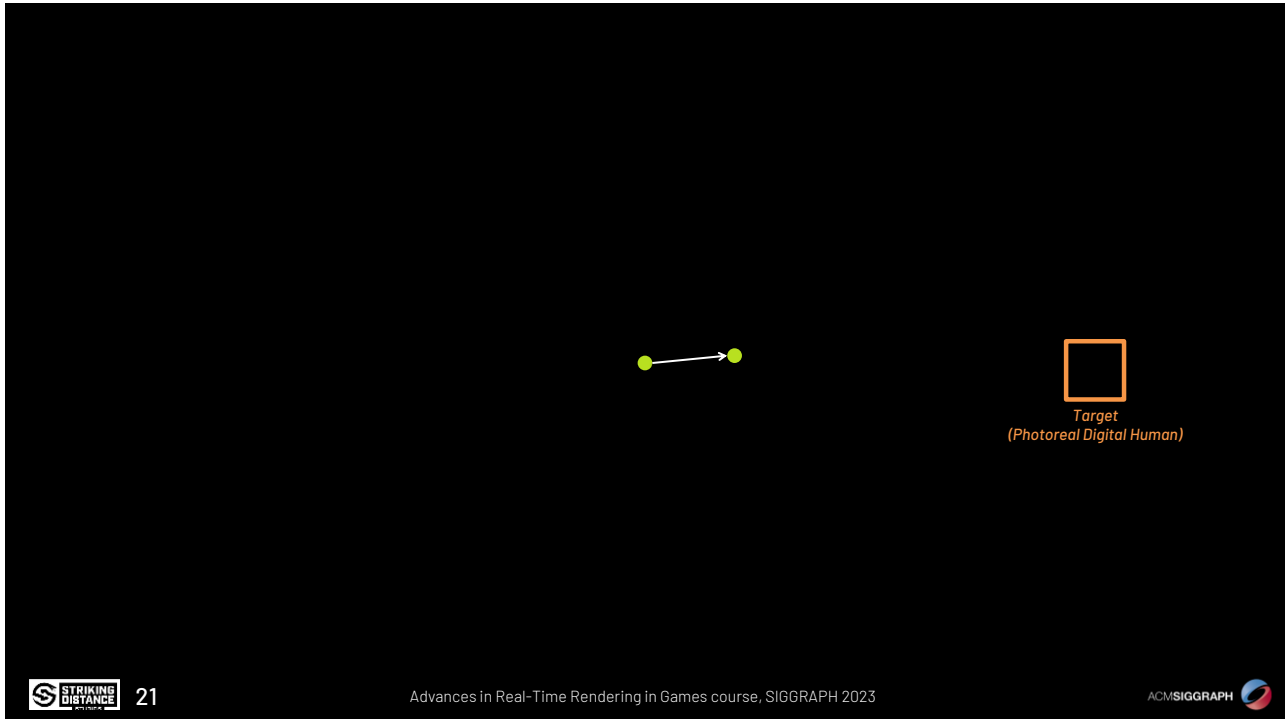
●
Start Position

Imagine we are authoring a digital human and the process is represented by a walk from this starting position...

Start Position

Target
(Photoreal Digital Human)

...towards a target.

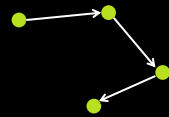


Every decision we make either gets us closer or further away from the target.



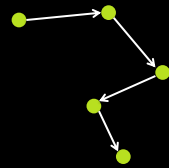

Target
(Photoreal Digital Human)

Sometimes we move forward.



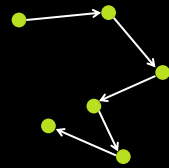

Target
(Photoreal Digital Human)

And sometimes the decisions are so unclear and difficult to make correctly that we can go backwards.




Target
(Photoreal Digital Human)

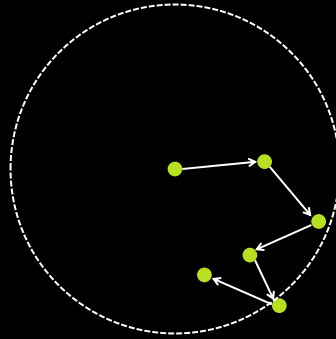
It is not difficult...




Target
(Photoreal Digital Human)

...to rapidly get back to square one.

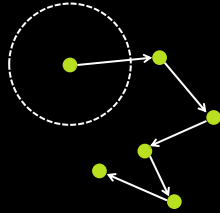
Perceived Differences an Untrained Person Can See




Target
(Photoreal Digital Human)

To make things more difficult, this is what most people will be able to see, or perceive, easily on images.

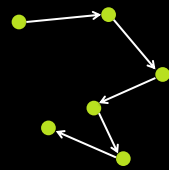
Perceived Difference We Need to See



Target
(Photoreal Digital Human)



But this is eye for photorealism you need. Smaller than the individual steps you have to take.



Target
(Photoreal Digital Human)

Statistically, it is difficult to find the solution as probabilities multiply at every decision.

And without a compass, escaping the maze is tremendously difficult.



A step can be:

Which gloss value to use.

How much subsurface we need.

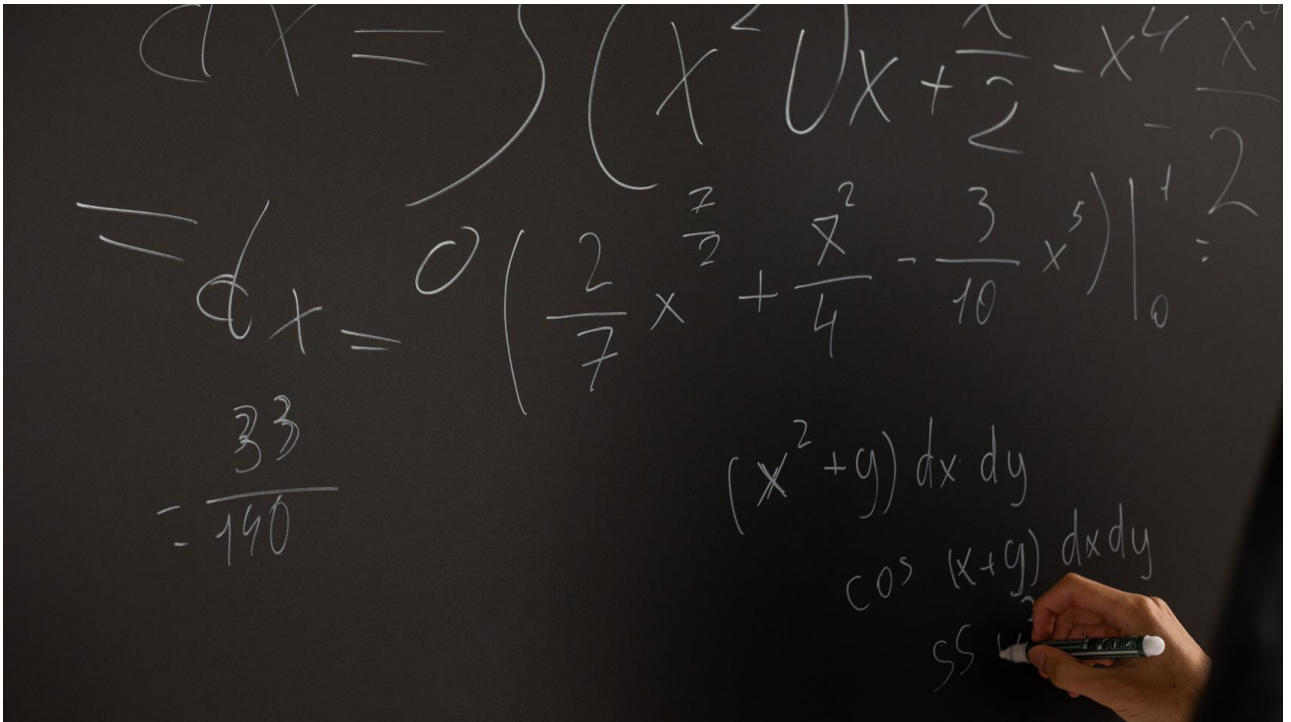
Or as simple as which skin color to use.



We often take concentric circles when trying to resolve a problem.



But in my experience, there is often a clear path to what has to be achieved.



But rather than taking it, sometimes we take side turns.

For example, the physics of the light transport are fascinating and important for rendering.

FROM PHOTONS TO MICROFACETS

Jorge Jimenez
Graphics R&D Technical Director



33

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023



And I personally heavily invested on it.

The more I understood the models we use to render, the more I realized how incomplete they are on certain regards.

So use the Fresnel Equations?

- More physically correct in principle
- More robust in practice
- More accurately model real-world substances
- Artistic control of edge behavior

Fresnel Equations

$$F(\theta) = \frac{F_s(\theta) + F_p(\theta)}{2}$$

$$F_s(\theta) = \frac{a^2 + b^2 - 2a \cos \theta + \cos^2 \theta}{a^2 + b^2 + 2a \cos \theta + \cos^2 \theta}$$

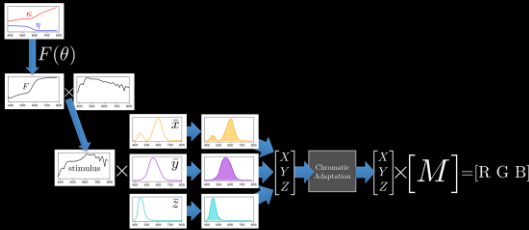
$$F_p(\theta) = F_s(\theta) \frac{a^2 + b^2 - 2a \sin \theta \tan \theta + \sin^2 \theta \tan^2 \theta}{a^2 + b^2 + 2a \sin \theta \tan \theta + \sin^2 \theta \tan^2 \theta}$$

$$2a^2 = \sqrt{(\eta^2 - \kappa^2 - \sin^2 \theta)^2 + 4\eta^2 \kappa^2} + (\eta^2 - \kappa^2 - \sin^2 \theta)$$

$$2b^2 = \sqrt{(\eta^2 - \kappa^2 - \sin^2 \theta)^2 + 4\eta^2 \kappa^2} - (\eta^2 - \kappa^2 - \sin^2 \theta)$$

The Schlick Approximation

$$F(\theta) \approx r + (1 - r)(1 - \cos \theta)^5$$



[Hoffman2019] Fresnel Equations Considered Harmful

Hoffman presented in 2019 how using the more accurate Fresnel equations could yield less accurate results than Schlick Fresnel, if we don't use them in a carefully precise and correct way.

There are multiple steps that have to be taken for Fresnel equations to be accurate.

And you need all of them to beat the Schlick Fresnel Approximation.



We can make a similar observation regarding approximations when trying to find the correct value to use for the reflectance of the eyes.

EYE REFLECTION

- Back and forth in previous projects, never settled down in a good value
- PBR predicts 2% using index of refraction of tear film
- Too low in practice

If you go by the book, and you take the index of refraction of the tear film, which is the most obvious choice, you get 2 percent.

That is too low reflectance in many situations.



Going further, I always wondered about what could be causing the sparkle in the eye.

Why when we are emotional the reflections on the eyes became brighter.

WHAT CAUSES THE SPARKLE OF THE EYE?

- Reflection is function of:
 - Specular (depends on IOR)
 - Roughness (tear film is a mirror)
- How can brightness of reflection increase with emotions?

How can that be explained using Physically based rendering?

Reflection is function of the specular reflectance and roughness.

The former depends on the index of refraction, which is a property of the medium and cannot change over time.

In that sense, it cannot fluctuate when we are happy, or sad.

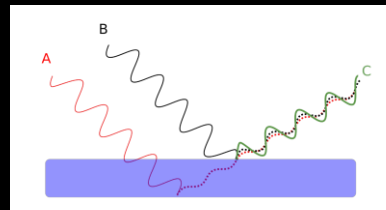
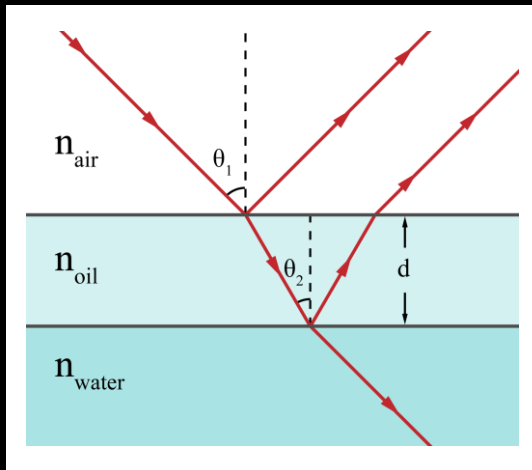
The roughness cannot change either, as the tear film is mirror-like on the cornea.



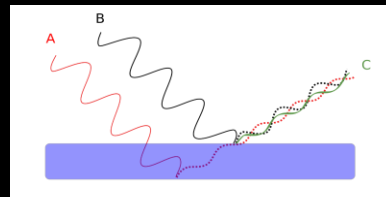
Surprisingly, both the eyes and soap bubbles have a common property.

They showcase a phenomena called thin film interference, which causes the color variations we see here.

THIN FILM INTERFERENCE



Constructive Interference



Destructive Interference

Thin film interference manifests when you have multiple mediums stacked with different indices of refraction.

In the case of this picture, we have air, oil and water. That causes light to bounce twice, first on the oil, and second in the water.

In brief, because of wave properties of the light, it can make reflection weaker or stronger.

It is a function of the thickness of this thin film, and depending on how much of this film we have in the eyes, the reflections will be more, or less visible.

Reflectometry of the precorneal film

Thomas Olsen

Department of Ophthalmology (Chief: Niels Finsen), University of Aarhus, Århus Kommunehospital, Århus, Denmark

ABSTRACT. In order to gain information about the thickness of the oily layer of the precorneal film, a clinical method of measuring the reflectivity of the precorneal film was developed. The method entails the use of a dichroic phenomenon to measure the reflectivity at a selected wavelength, 500 and 700 nm, at 27 incidence angles. Based on a physical model of the oily layer acting as a thin dielectric film, a theory is given on the interpretation of the results in terms of the thickness of the oily layer. In normal subjects a mean reflectivity of 0.79% (SD 0.63) and 3.16% (SD 0.79) was found at 500 and 700 nm, respectively. A significant positive correlation between the measurements at the 2 wavelengths was found. It is assumed, that these results are consistent with a thickness of about 60 nm, as being the most probable thickness of the oily film. This value seems to be in accordance with other experimental data. Objective measurements may provide an important new tool for the study of the anterior part of the ocular surface.

Key words: cornea - oily film - oily layer - reflectivity measurements - photometry

Materials and Methods

The precorneal system was similar to that described for *in vivo* measurements of corneal scatter by the lamp photometer (Olsen 1982). Briefly, this method employs a fiber optic probe (diameter: 150 µm) incorporated into the eyepiece of the slit lamp, by means of which light is picked up in a small area of the image plane and guided into a photomultiplier (Fig. 1). In the present study a magnification in the image plane of the eyepiece of 3.1 X was used, so that the projected diameter of the optic probe to the object plane amounted to 463 µm.

The reflectivity measurements were made with an incidence angle of 30° (40° between slit-light and microscope) on the central precorneal film. As

Table 1.
Reflectivity of the precorneal film in 10 normal subjects. The 95% confidence limits were calculated for the average values according to the formula $\bar{x} \pm t_{0.05} \cdot sD/\sqrt{n}$.

| No. | Sex | Age | 500 nm | 700 nm |
|-----------------------|-----|-----|------------|------------|
| 1 | F | 40 | 3.94% | 2.65% |
| 2 | M | 35 | 3.60% | 2.26% |
| 3 | F | 44 | 5.31% | 3.25% |
| 4 | F | 34 | 3.25% | 2.43% |
| 5 | F | 36 | 5.13% | 4.52% |
| 6 | M | 31 | 2.66% | 2.58% |
| 7 | M | 24 | 3.70% | 3.20% |
| 8 | F | 33 | 3.82% | 3.79% |
| 9 | M | 33 | 4.52% | 2.69% |
| 10 | M | 32 | 4.62% | 4.27% |
| Mean | | | 4.06% | 3.16% |
| SD | | | 0.83% | 0.79% |
| 95% confidence limits | | | 3.47-4.65% | 2.60-3.72% |

95% confidence limits 2.43-4.62% 3.00-2.58%
SD 0.94% 0.56%
95% 3.96% 2.16%



This is known since the eighties in the medical field, where they measure the reflectance, and use inverse rendering to infer the thickness of this precorneal film.

PBR VS. REALITY

- Without thin film interference we cannot:
 - We cannot infer correct regular reflectance (2.5 - 5.5%)
 - Effectively model emotions through the eyes (4%+ reflectance)
- Incomplete physical models used rigidly are Dangerous

What I learnt from this, is that physical models can be dangerous if we take them as the ground truth.

Specially if we are missing important components, such as in this case, thin film interference.

If you want to base exclusively on physical models, then you must model every single nuance of the rules defined by physics.

WORKING WITH ROBUST REFERENCES



When working in new BRDFs, my proposal is to always validate with real measured data.

To make sure that, like in the case of the Fresnel equations, every step we do is getting us closer to photographs, which is one of the main goals of using those physical models.

In other words, the target for us is not to achieve more correct physically based rendering results, but to get closer to a photograph.

This sequence of images that we see on this slide represent the ultimate reference for photorealism.

OLAT, or one light at a time. A subject or object captured under multiple light and view directions. This defines the target we rally around.

IMPLICIT VS EXPLICIT BRDF BEHAVIOUR

- Implicit
 - Assume our model is correct
 - Input index of refraction and roughness
 - Everything else automatically driven by physics
- Explicit
 - Recognize our models are not universal and still not complete
 - Expose explicit controls to drive visual features

It has been a trend to drive parameters automatically with physically based models, which show implicit behaviors.

For example, driving retroreflection or Fresnel properties using roughness.

If we acknowledge that real-time physical models are, as of today, uncomplete, we could opt instead for a more explicit behavior.

Giving us today, the control required to match photographic reference.

This line of thought was the foundation of our approach to materials.

ITERATING MATERIALS

SWITCHING CAMERA / LIGHTS
FOR VALIDATION

CORRECT VISUALIZATION

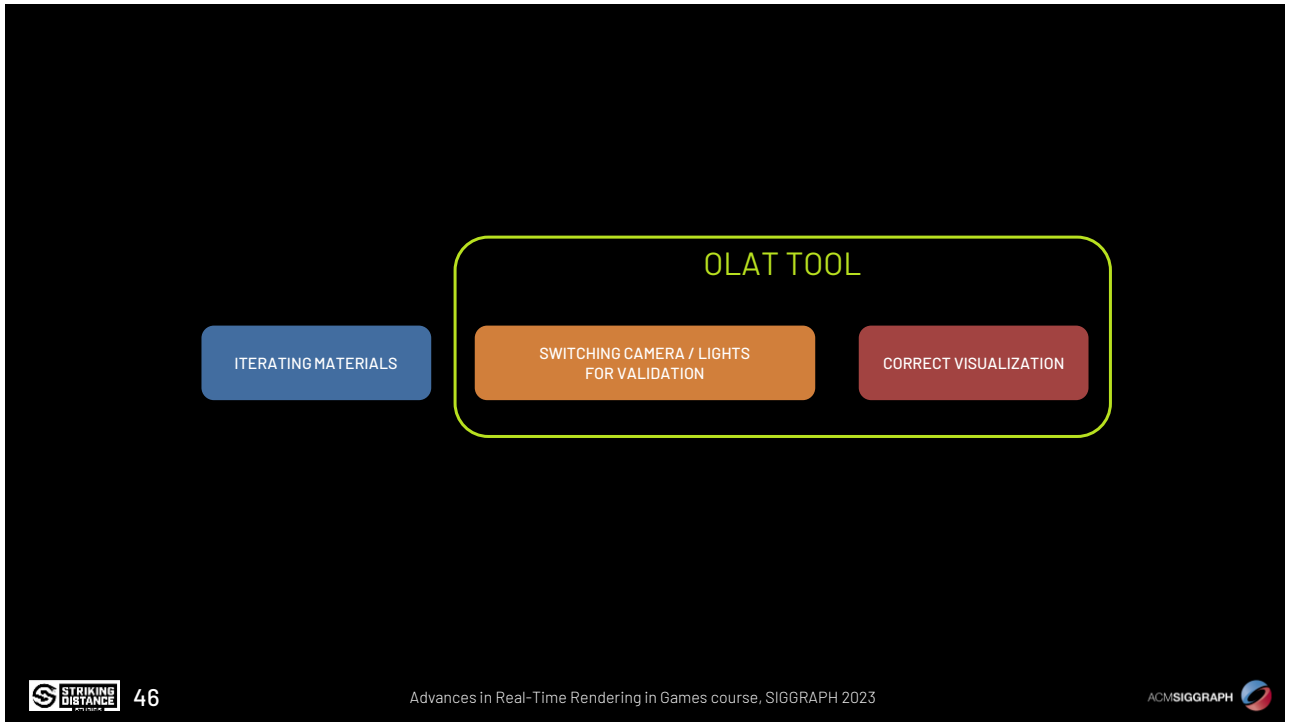
With that target defined, now the challenge is to accelerate the processes to create digital doubles.

First problem is how to iterate materials fast, which we covered in our Character Rendering Art talk in GDC.

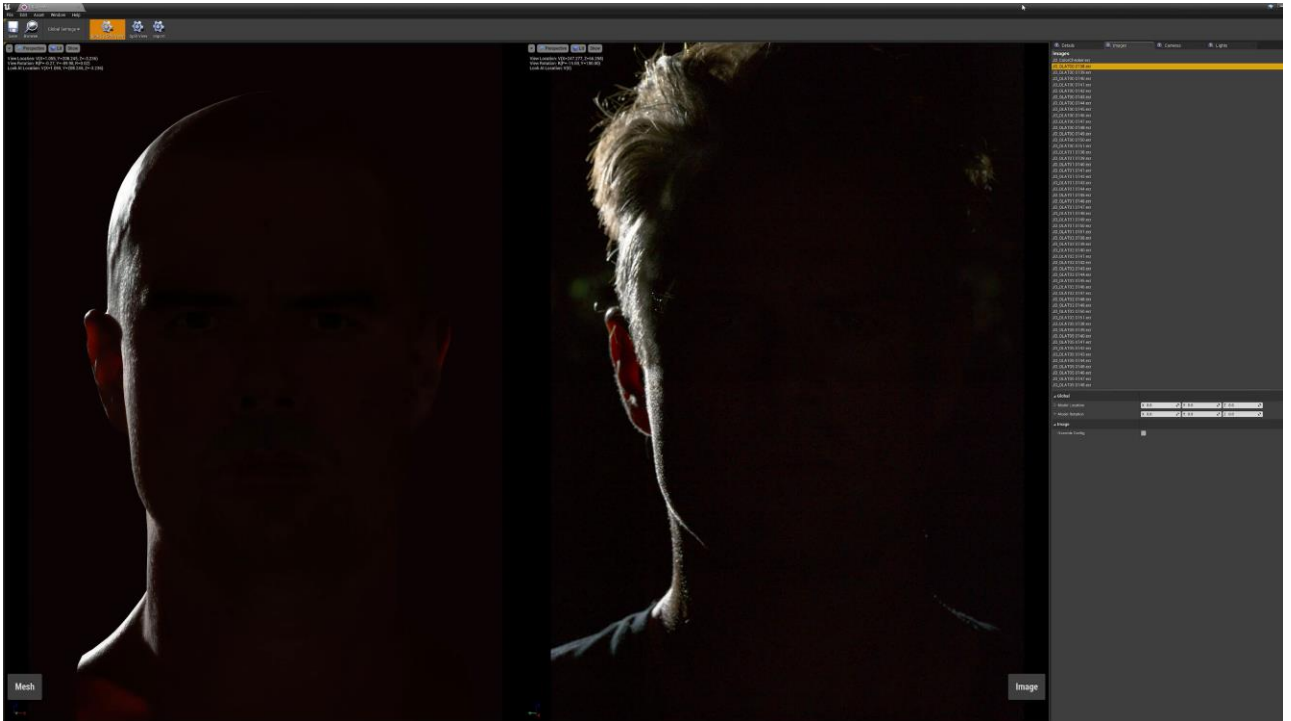
Second is The time it takes to constantly switch lights for validating different light or view directions.

Third is that we cannot do a side by side in different software, as the reference OLAT images are in linear space, but the engine is tonemapping the colors.

That is not apples to apples.



For that, we built the OLAT Tool.



This tool allows us to compare with measured data rapidly.

Switching lights or cameras is a click or even, a hotkey away.

This allows working under multiple lighting conditions much faster.

We will see this tool during the presentation, as it was a key element for us to observe and learn about photorealism.

SMALL SCALE

LARGE SCALE

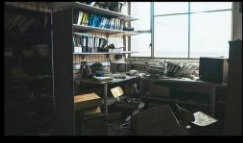
BRDF



PATCH



CLUTTER



CORRIDOR



STREET



MATERIALS

LIGHTING



48

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023

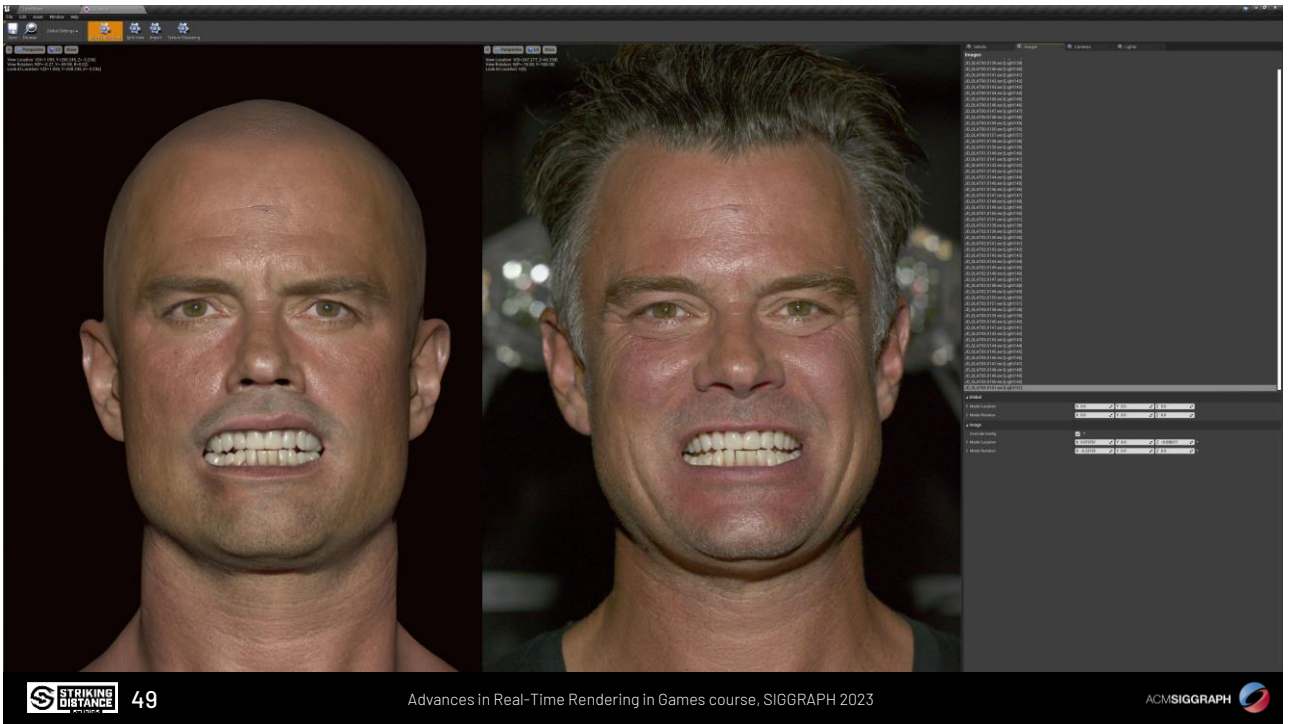


I like to think of photorealism as a problem that we can divide in scales.

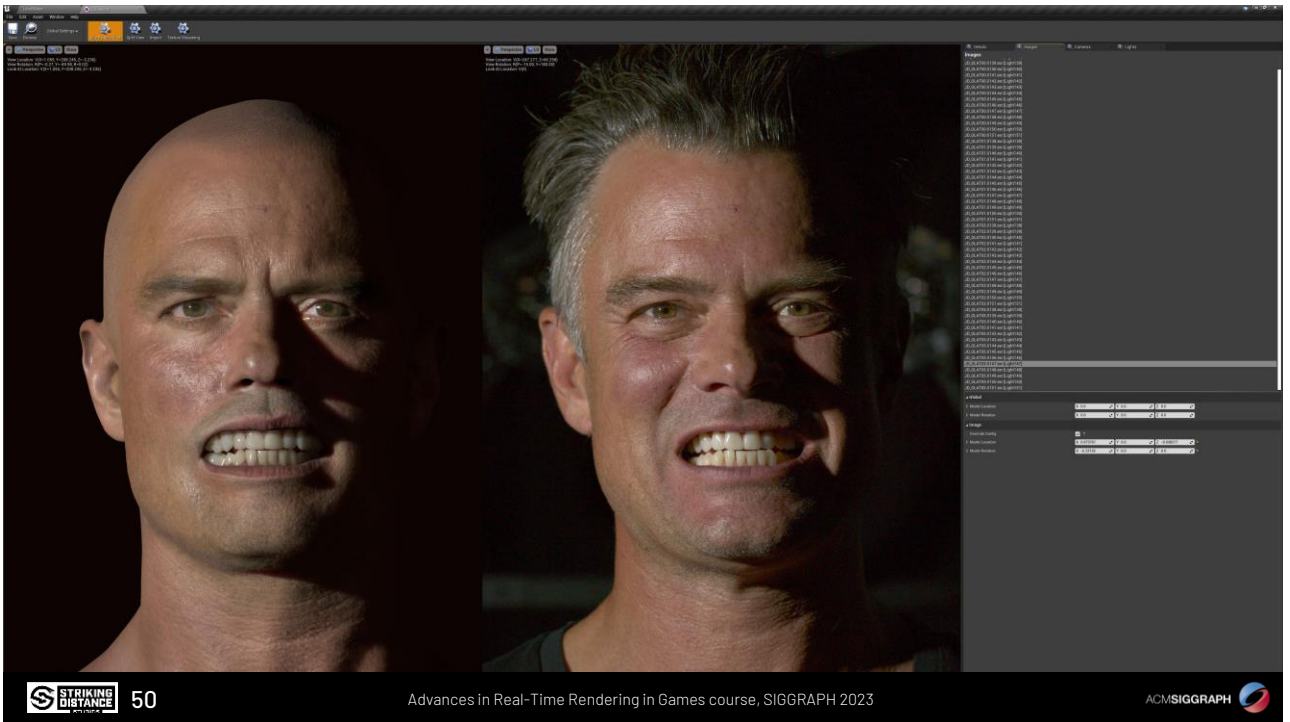
And all of them matter.

From the small scale properties defined at the material level, to the larger scaler detail that lighting provides.

I will start with how we approached designing BRDFs to better match photographic reference.



To motivate the problem, here we are matching the teeth appearance when lighting from the front.



If we move the light to the side, the teeth is too dark.

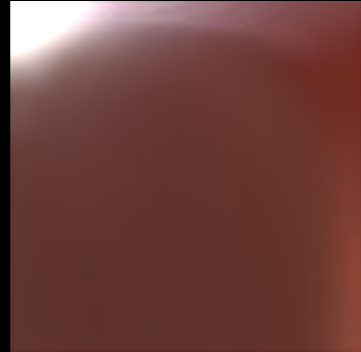
There are no values in the textures that would help us with this, as this is a problem in the BRDF.

Material consistency over light incidence and view angles.

This led us to this goal.

Consistency over light incidence and view angles.

BRDF SLICE



Example BRDF Slice

That consistency can only be achieved with a tool called BRDF.

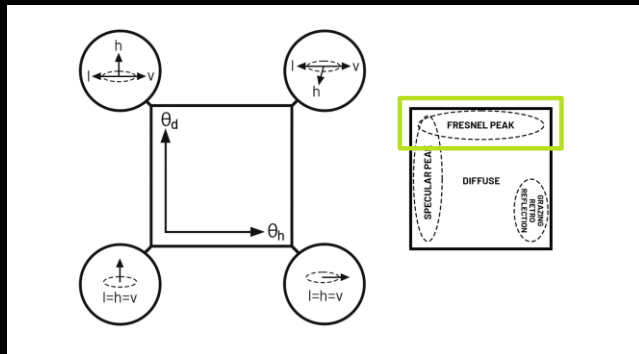
A BRDF tells us how much a material reflects at a given light incidence angle and outgoing view direction.

A useful way to look at BRDFs is through a BRDF slice visualization.

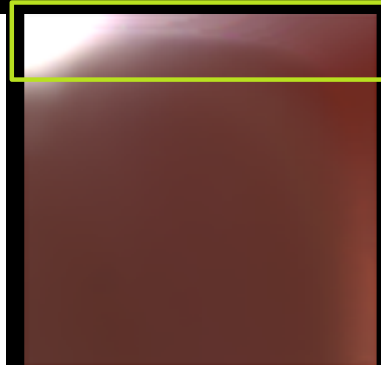
The inputs are the vertical and horizontal axes. The output is the value in the image.

The brighter, the more light it reflects. It has four important regions.

BRDF SLICE



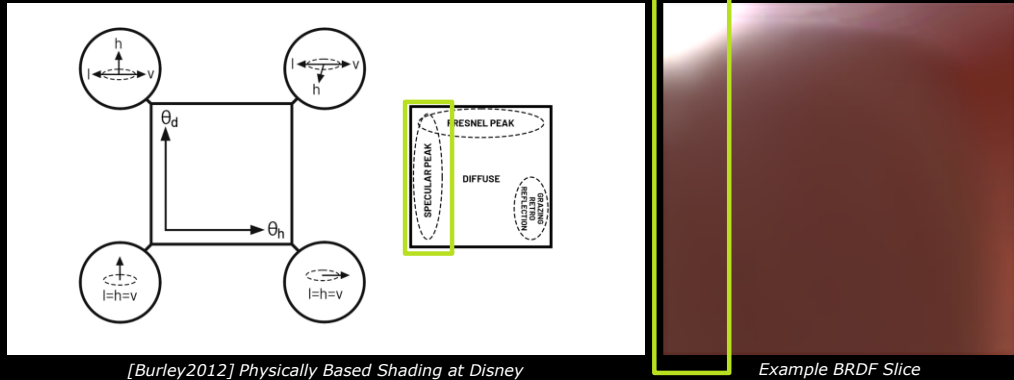
[Burley2012] Physically Based Shading at Disney



Example BRDF Slice

The Fresnel peak, located on the top side of the image, which defines what happens when you backlit an object.

BRDF SLICE

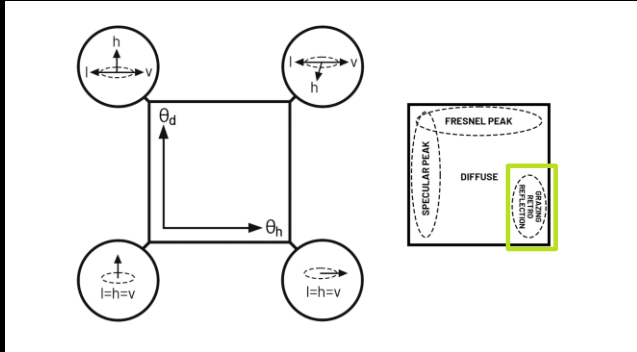


[Burley2012] Physically Based Shading at Disney

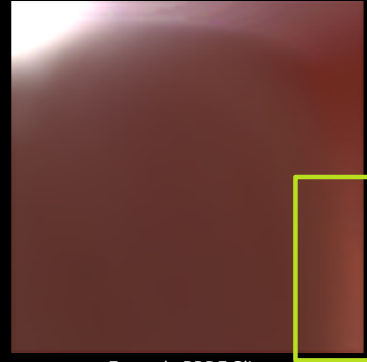
Example BRDF Slice

The specular peak, located on the left side of the image, which defines the area of maximum specular reflection.

BRDF SLICE



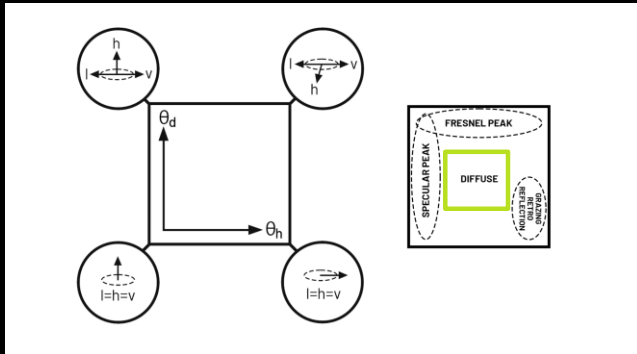
[Burley2012] Physically Based Shading at Disney



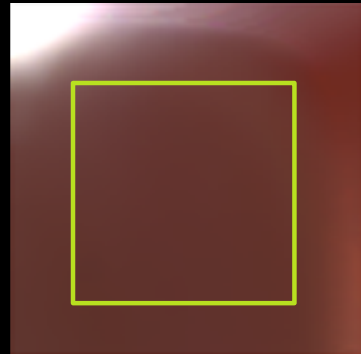
Example BRDF Slice

The retroreflection, which is how much light comes back at you when you front lit an object but the surface points to the sides.

BRDF SLICE



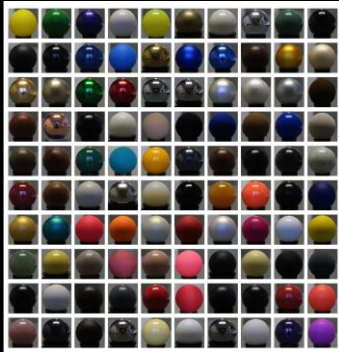
[Burley2012] Physically Based Shading at Disney



Example BRDF Slice

Finally, the diffuse reflection, which covers everything else.

MERL DATABASE



[Matusik2003] A Data-Driven Reflectance Model

New Controls

MATCHES PHOTOGRAPHS?

--- Pragmatic

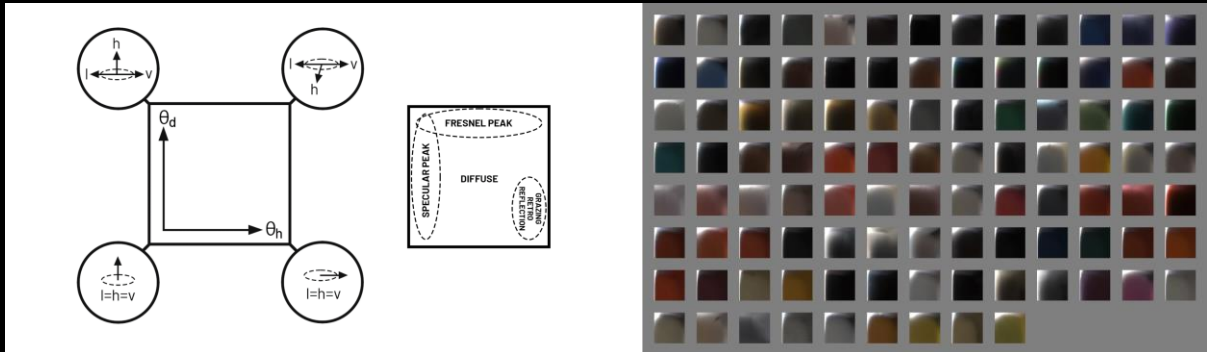
BEHAVIOUR IN MERL DATABASE?

--- General

Now let me introduce the MERL database, which contains captured BRDF data for 100 different materials, including plastics, metals and cloth.

We used this database to make sure the controls we added to match the reference photography would also add behaviors exhibited in this database, to ensure generality outside of our use cases.

MERL DATABASE



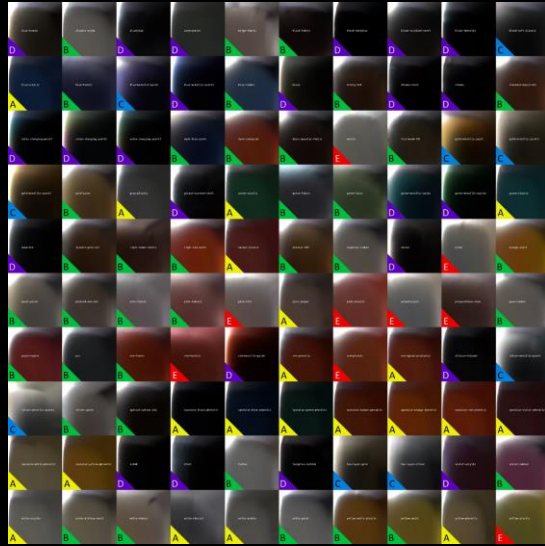
MERL Database BRDF Slices

For each of the materials in this database we can draw its BRDF slice.

There is a variety of differences on them, and some models like Oren Nayar or Burley, approximated the appearance of some of them through implicit behavior.

But to match a photograph we found useful to have finer control than that.

MERL DATABASE CLASSIFICATION



We classified the materials in this database into categories that exhibit similar characteristics.

A, B, C, D and E.

LOOKING FOR PATTERNS



Now if we take them apart, we can see that they look different but like previous work also found, they have some common characteristics.

They brighten or darken in the top and right side of the image.

Sometimes they have a darker top right corner.

They are Fresnel, retroreflection and smooth terminator effects. This is exactly what we also found in our own measurements when comparing regular shading models against photographs of humans.

BRDF EXPRESIVENESS

- [Lambert1760]
 - Assumes Lambertian radiator
 - No expressiveness

To model this, we are looking for expressiveness in the BRDF design.

Lambert, the gold standard, and longstanding for centuries cannot model these subtleties.

BRDF EXPRESIVENESS

- [Lambert1760]
 - Assumes Lambertian radiator
 - No expressiveness
- [Oren-Nayar1994]
 - Microfacets with Lambertian behavior
 - V-Cavity assumption

Oren Nayar, can show retroreflection and some Fresnel properties, dependent on roughness.

BRDF EXPRESIVENESS

- [Lambert1760]
 - Assumes Lambertian radiator
 - No expressiveness
- [Oren-Nayar1994]
 - Microfacets with Lambertian behavior
 - V-Cavity assumption
- [Burley2012]
 - Follows behavior observed on Merl Database
 - Self-driven visual features based on roughness

Burley accurately represents materials of the MERL database but also has an implicit behavior based on the roughness.

BRDF EXPRESSIVENESS

- [Lambert1760]
 - Assumes Lambertian radiator
 - No expressiveness
- [Oren-Nayar1994]
 - Microfacets with Lambertian behavior
 - V-Cavity assumption
- [Burley2012]
 - Follows behavior observed on Merl Database
 - Self-driven visual features based on roughness
- We found existing models not expressive enough to match photographic reference

If we adjust the roughness for the speculars, sometimes the diffuse appearance we get is not what we are looking for.

Possibly because in complex materials like human skin, there is a composition of layers that interact with each other, which cannot be described with simple implicit rules.

Can a simple pragmatic model express reality?

Now our question was, can we use something simple that will still allow us to match photographs?

Not really paying attention to what statistics predict but something expressive and general that we can use to directly match behavior that we observe.

CALLISTO BRDF

With this goal in mind we created the Callisto BRDF.

CALLISTO BRDF: DESIGN PRINCIPLES

- Create a universal and expressive BRDF
 - Future proof and capable of representing any material outside of MERL
 - Avoiding magic numbers in the math other than to facilitate range sanity
 - Using simple arithmetic
 - Using orthogonal parameters
 - Pursuing an empirical visual match rather than physics
- Artist and Machine Learning friendly
- Default values match Lambert + GGX
- Perform well against the MERL database and in our digital doubles (OLAT)

67

We followed a set of rules for that.

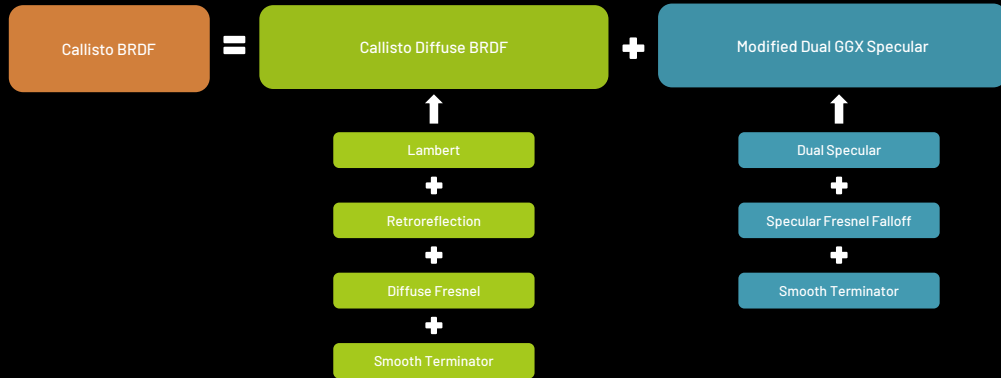
First, we wanted the BRDF to be universal and expressive, avoiding magic numbers, and using simple arithmetic.

Second, we wanted it to be useful for artists, with few parameters, but also future proof and machine learning friendly.

Third, we wanted the default values to match the industry standard, which is Lambert and GGX.

Finally, we wanted it to perform well both in the MERL database and in our own digital doubles.

CALLISTO BRDF



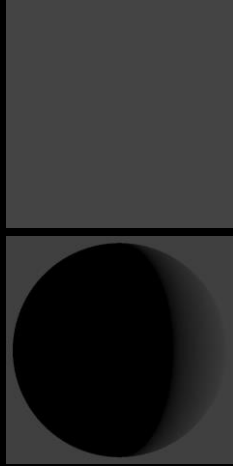
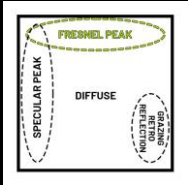
Like most BRDFs, it has two components. Diffuse and specular.

For diffuse we use Lambert as baseline, but with configurable retroreflection and Fresnel elements.

For speculars we use dual ggx, with new Fresnel controls.

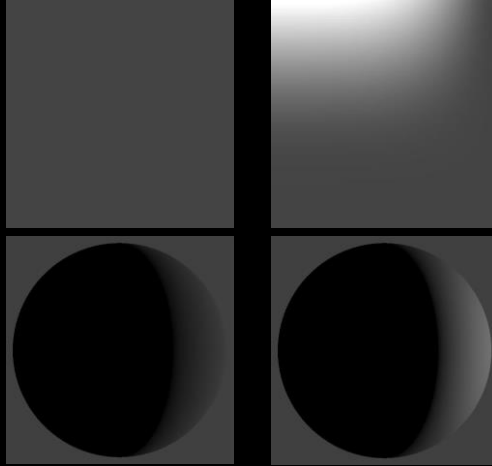
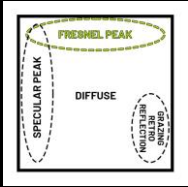
Finally, a smooth shading terminator is applied to both the diffuse and specular components.

CALLISTO BRDF: DIFFUSE FRESNEL



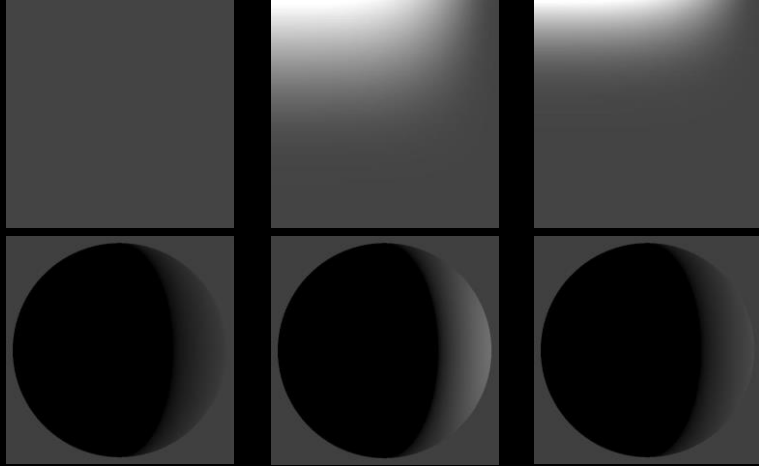
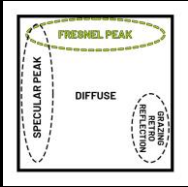
Lets dive first into the diffuse Fresnel controls.

CALLISTO BRDF: DIFFUSE FRESNEL

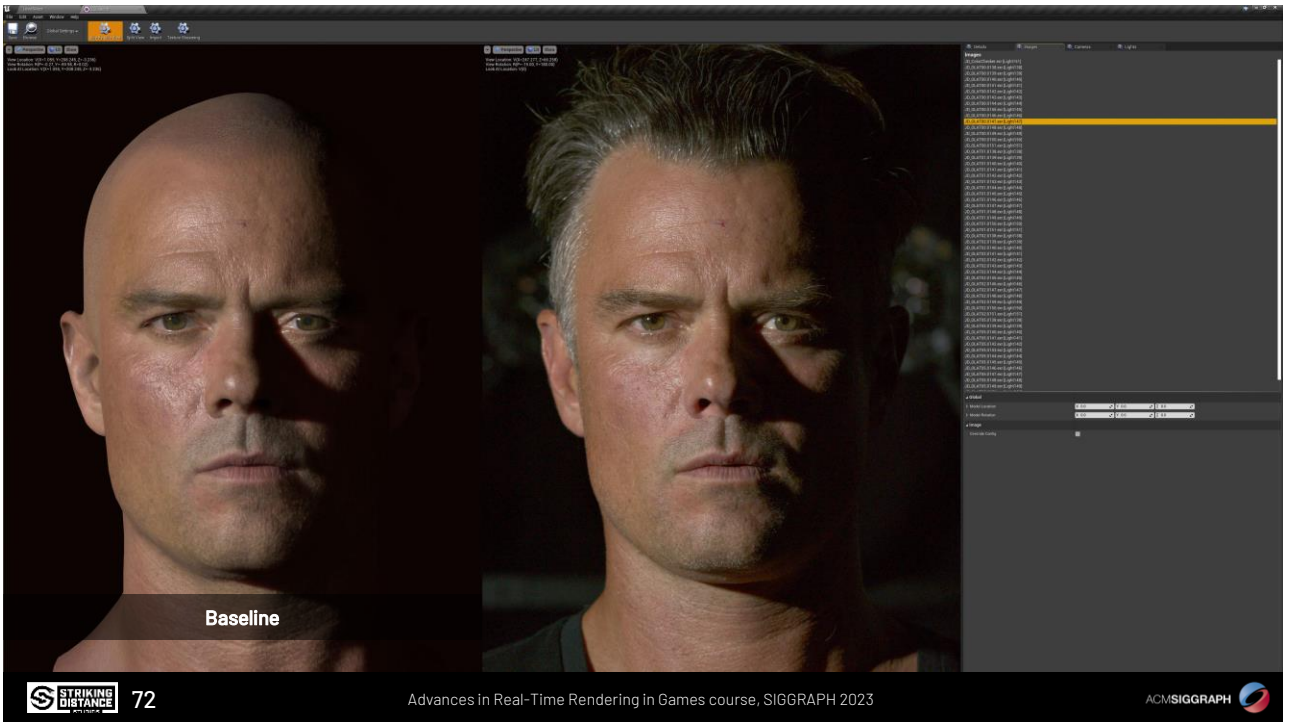


They allow to brighten or darken the Fresnel response of the diffuse to taste.

CALLISTO BRDF: DIFFUSE FRESNEL

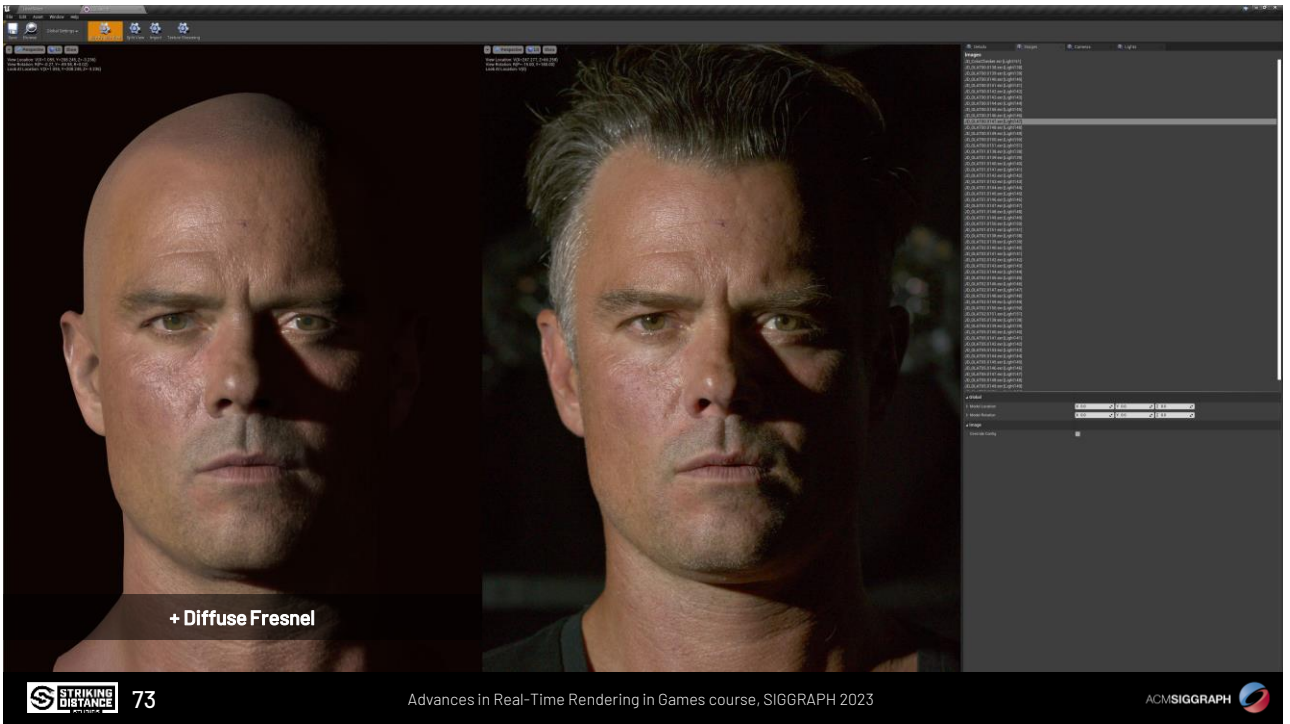


And to control how fast that will happen with a falloff.

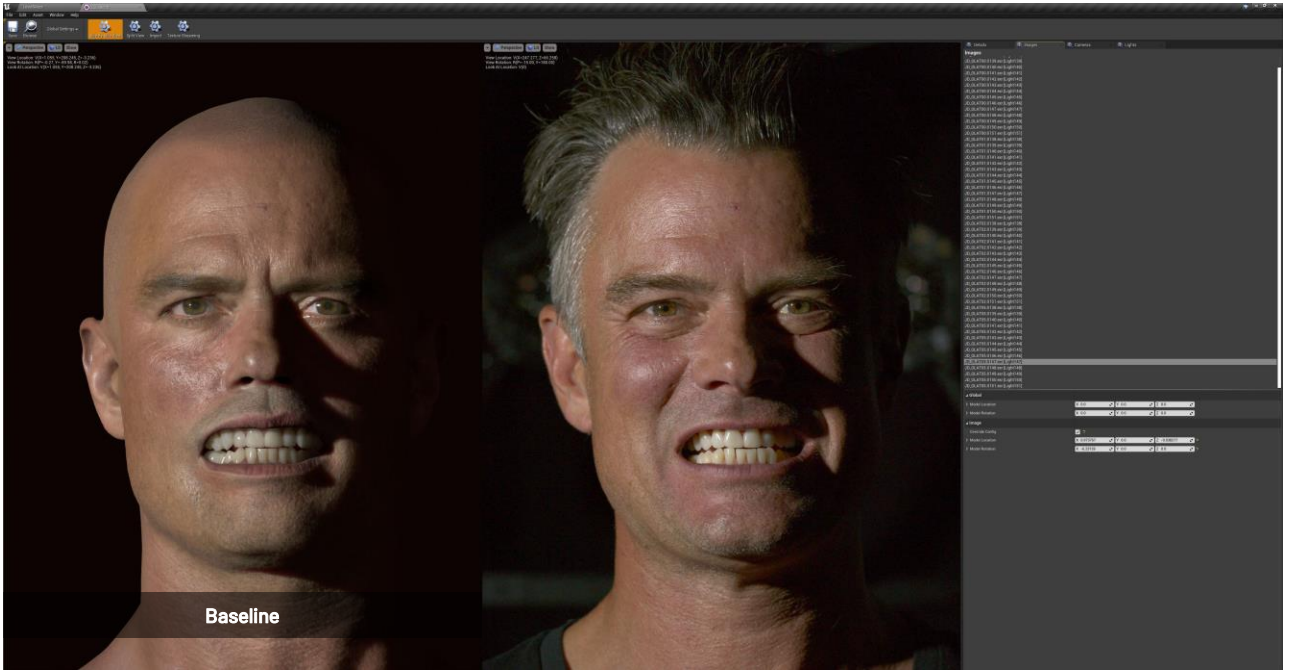


We found these controls to be important teeth and eyes.

If we match the albedo of the eyes for frontal lighting, when we are side lighting, they will look too dark, as we can see in this image.



The new diffuse Fresnel controls allow to correct for this, while preserving the look when lighting from the front.



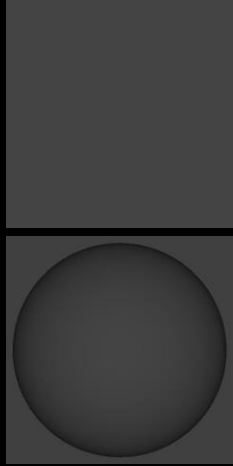
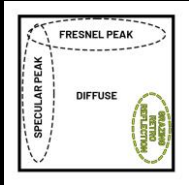
Baseline



+ Diffuse Fresnel



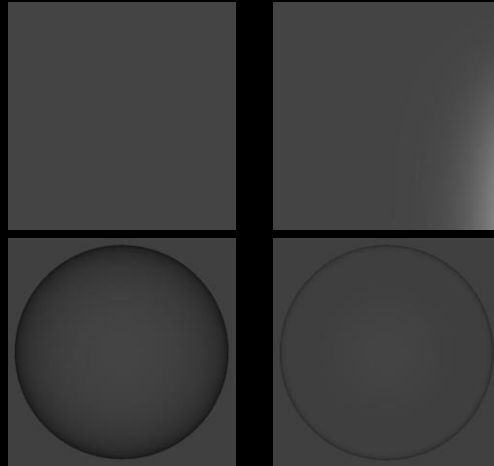
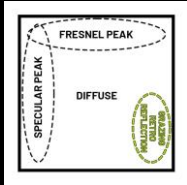
CALLISTO BRDF: RETROREFLECTION



We added symmetrical controls for the retroreflection.

CALLISTO BRDF: RETROREFLECTION

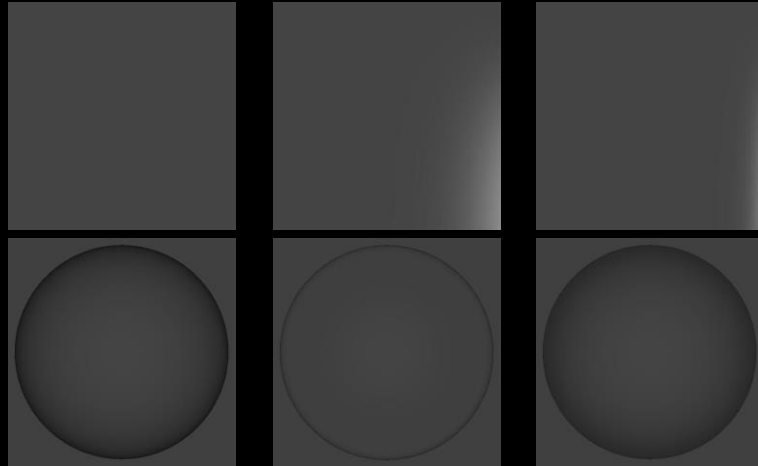
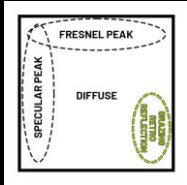
Retroreflection Controls



So we can adjust the reflection on the right side of the BRDF slice.

CALLISTO BRDF: RETROREFLECTION

Retroreflection Controls



And also control the falloff over which this happen.

The retroreflection controls allow to flatten or enhance the look of an object when lit and observed from the front.

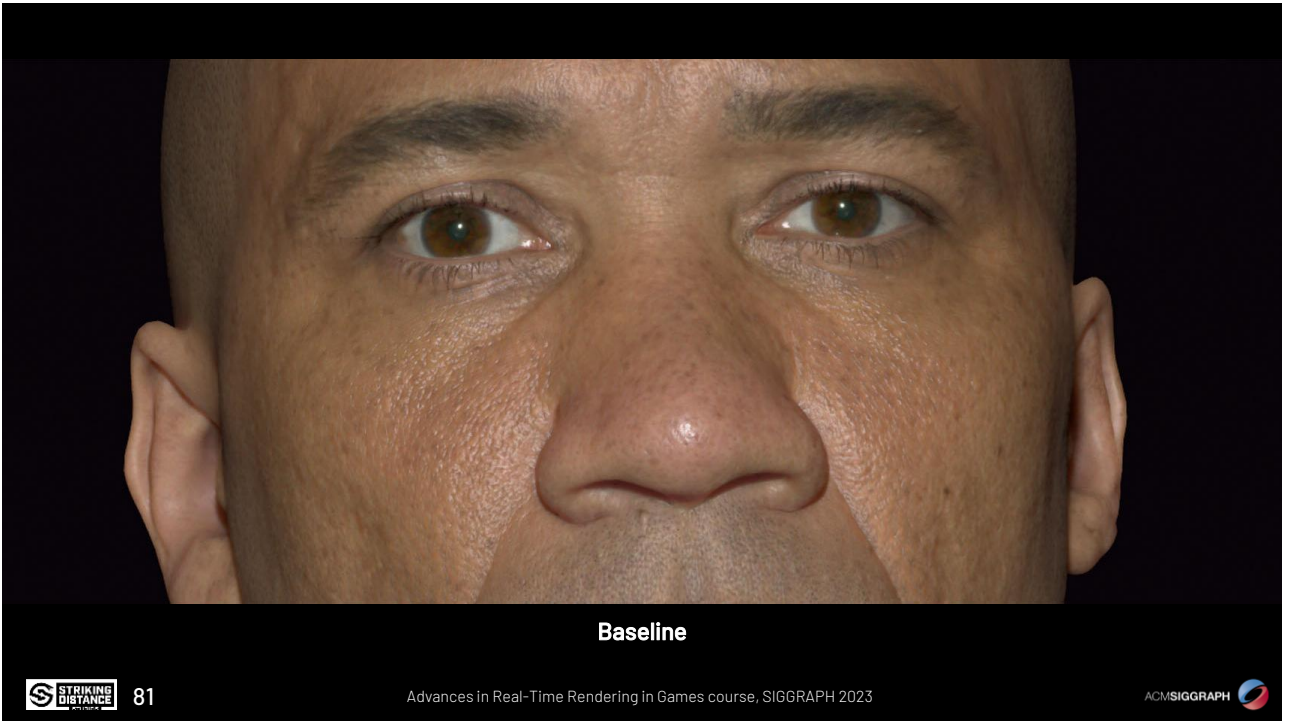


Retroreflection is important for the teeth and skin.

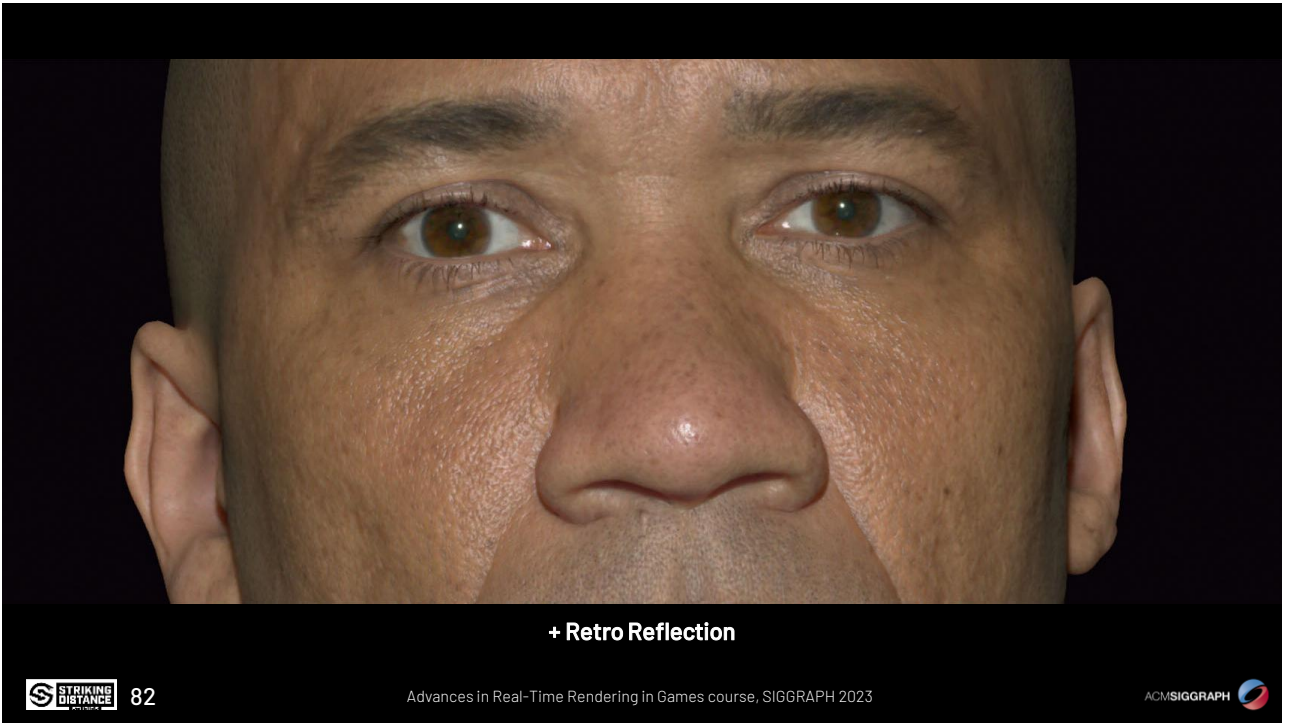
In this image we can see how it allows to render the green hues visible in the silhouette of human skin.



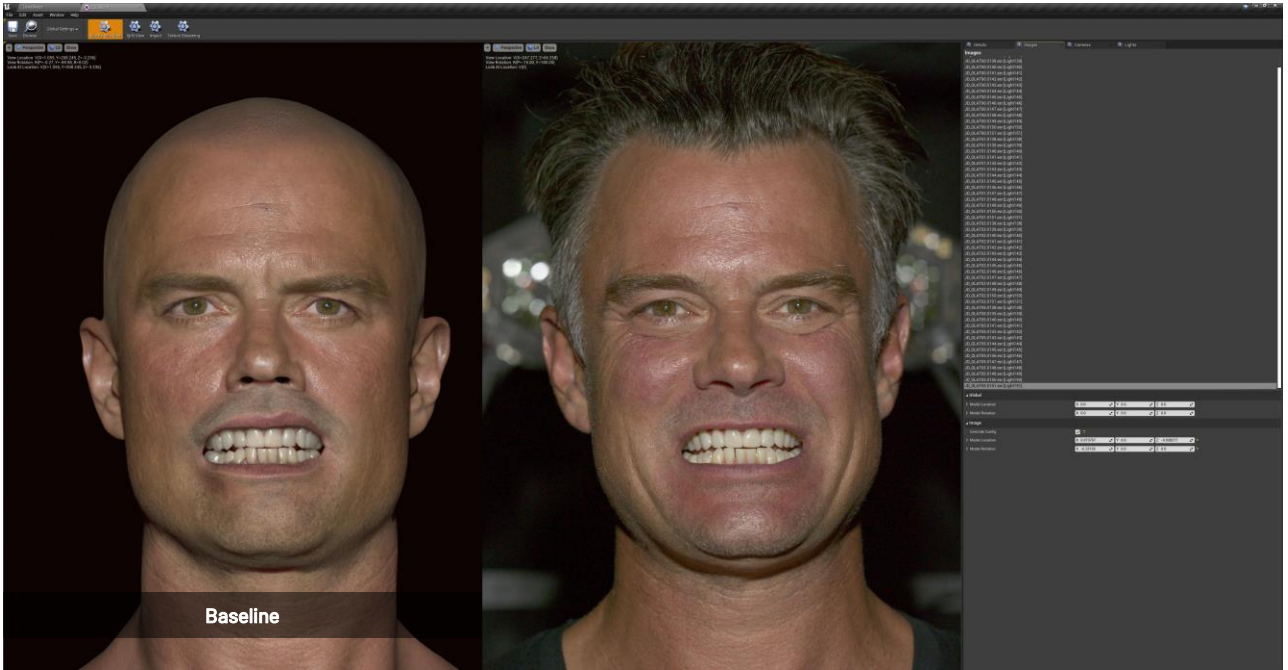
[back and forth]



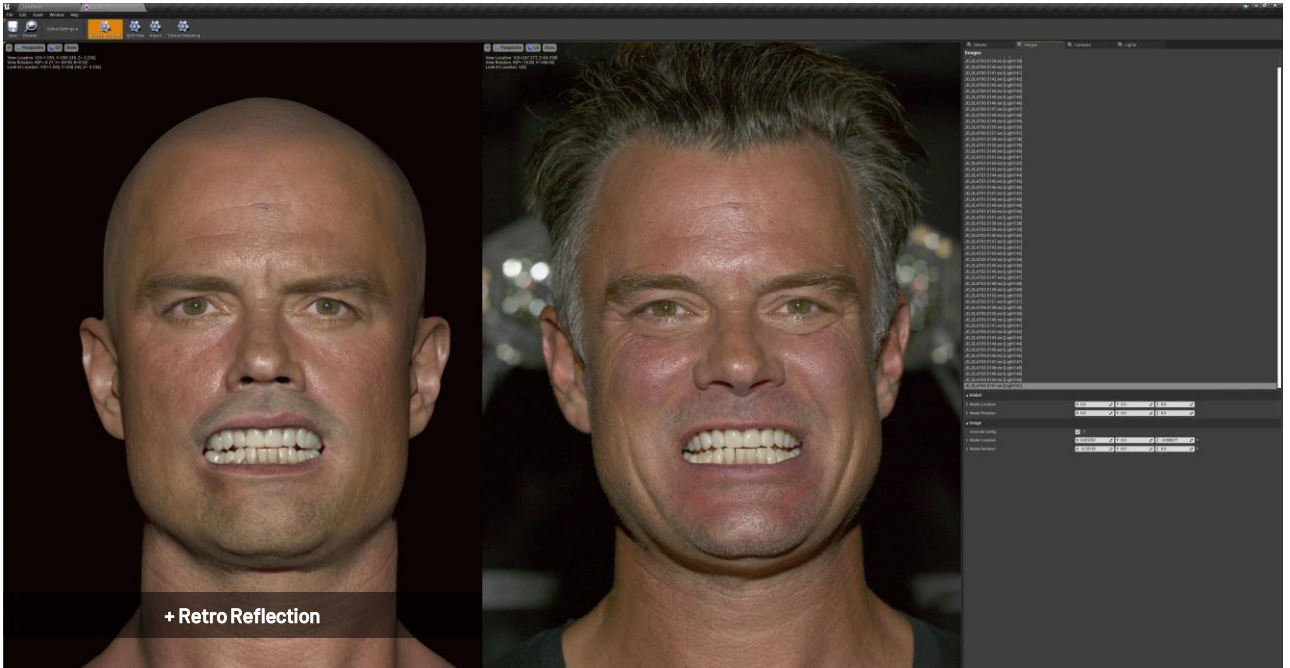
Now again but taking a closer look.



[back and forth]



Baseline



+ Retro Reflection



CALLISTO BRDF: FRESNEL AND RETROREFLECTION

LAMBERT + GGX BRDF

$$L_o = \left(f_{\text{lambert}}(\omega_i, \omega_o) + f_{\text{ggx}}(\omega_i, \omega_o) \right) L_i \overline{\cos \theta_i} d\omega_i$$

CALLISTO BRDF

$$L_o = \left(c_1(\theta_d, \theta_h) f_{\text{lambert}}(\omega_i, \omega_o) + f_{\text{ggx}}(\omega_i, \omega_o) \right) c_2(\theta_i) L_i \overline{\cos \theta_i} d\omega_i$$

$$c_1(\theta_d, \theta_h) = \underbrace{l(1, \rho_f, \alpha_f)}_{\text{Fresnel}} \underbrace{l(1, \rho_r, \alpha_r)}_{\text{Retroreflection}}$$

$$h(\theta, n, \varphi, m) = (1 - \overline{\cos \theta})^{5n} \overline{\cos \varphi}^{5m}$$

$$\alpha_f = h(\theta_d, r(n_r), \theta_h, r(m_r))$$

$$\alpha_r = h(\theta_h, r(n_r), \theta_d, r(m_r))$$

$$\tilde{x} = \max(x, 0)$$

$$r(x) = 2(1 - x)$$

$$l = \text{lerp}$$

$$f_{\text{lambert}}(\omega_i, \omega_o) = \rho / \pi$$

| | Range | Default |
|---------------------------------------|----------|---------|
| ρ_r Retroreflection* | [0, 256] | 1 |
| n_r Retroreflection Falloff | [0, 1] | 0.75 |
| m_r Retroreflection Tangent Falloff | [0, 1] | 0.75 |
| ρ_f Diffuse Fresnel* | [0, 256] | 1 |
| n_f Diffuse Fresnel Falloff | [0, 1] | 0.75 |
| m_f Diffuse Fresnel Tangent Falloff | [0, 1] | 0.75 |

* Exposed as intensity multiplied by a tint



85

(Developed with Jose Naranjo and Miguel Rodríguez)

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023

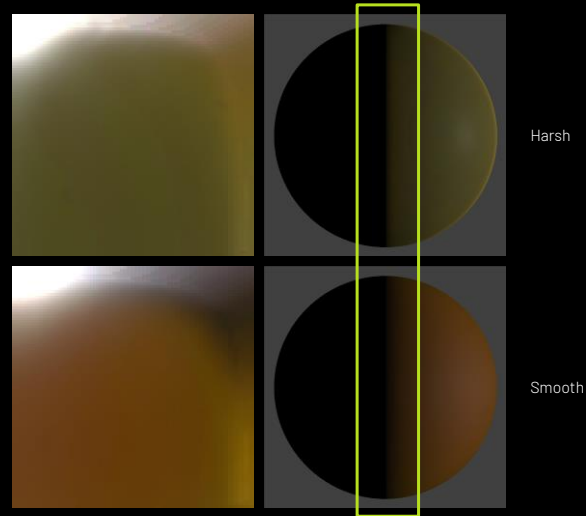
It was designed using the simplest mathematical building blocks.

Hence the mathematical definition of this is completely straightforward, so I won't get into the detail.

We share common grounds with Burley, with the main difference being avoiding implicit behavior and instead exposing more parameters by design.

(Note that the mathematical expression of the Callisto BRDF has slightly evolved since our prior GDC 2023 presentation)

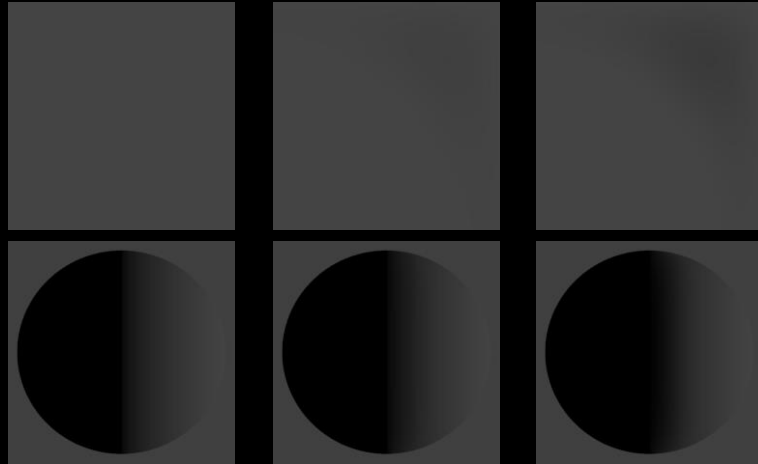
CALLISTO BRDF: SMOOTH TERMINATOR



We also found the Lambert shading terminator to be sometimes too harsh compared to reality.

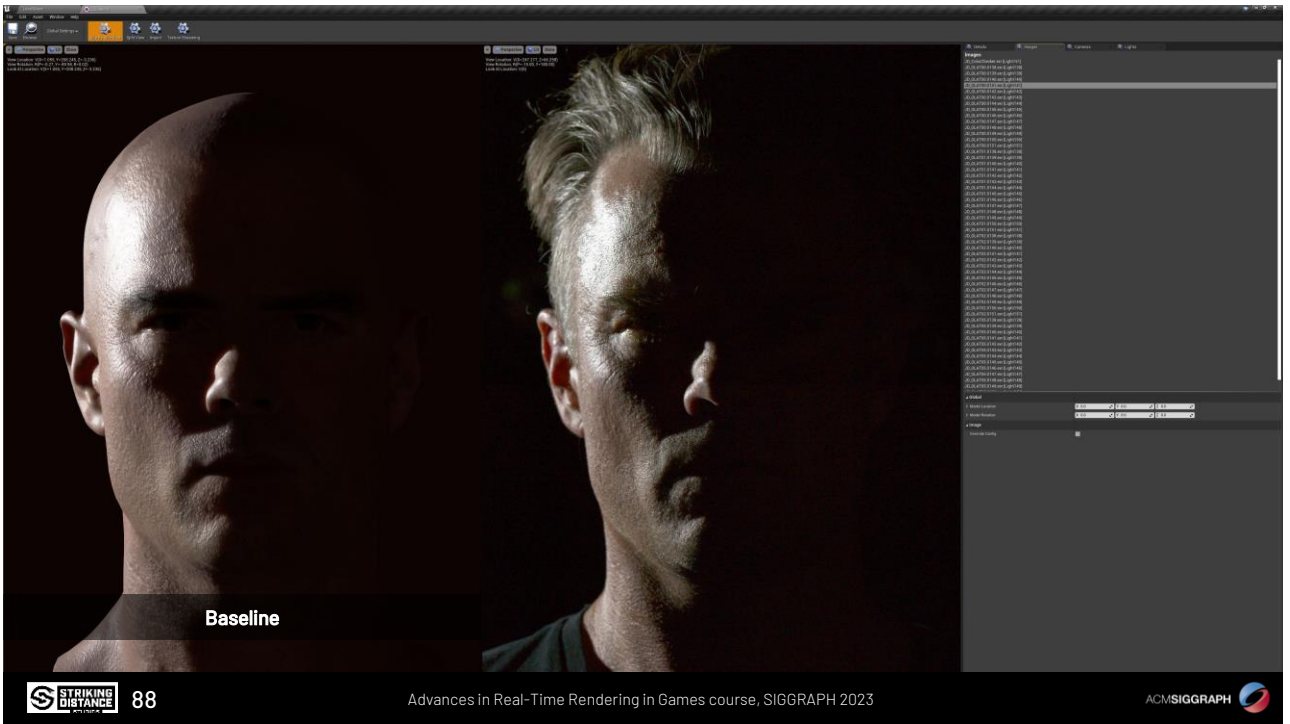
Looking at the MERL database we found evidence of materials that had a much smoother transition than Lambert, like in the example in the bottom.

CALLISTO BRDF: SMOOTH TERMINATOR



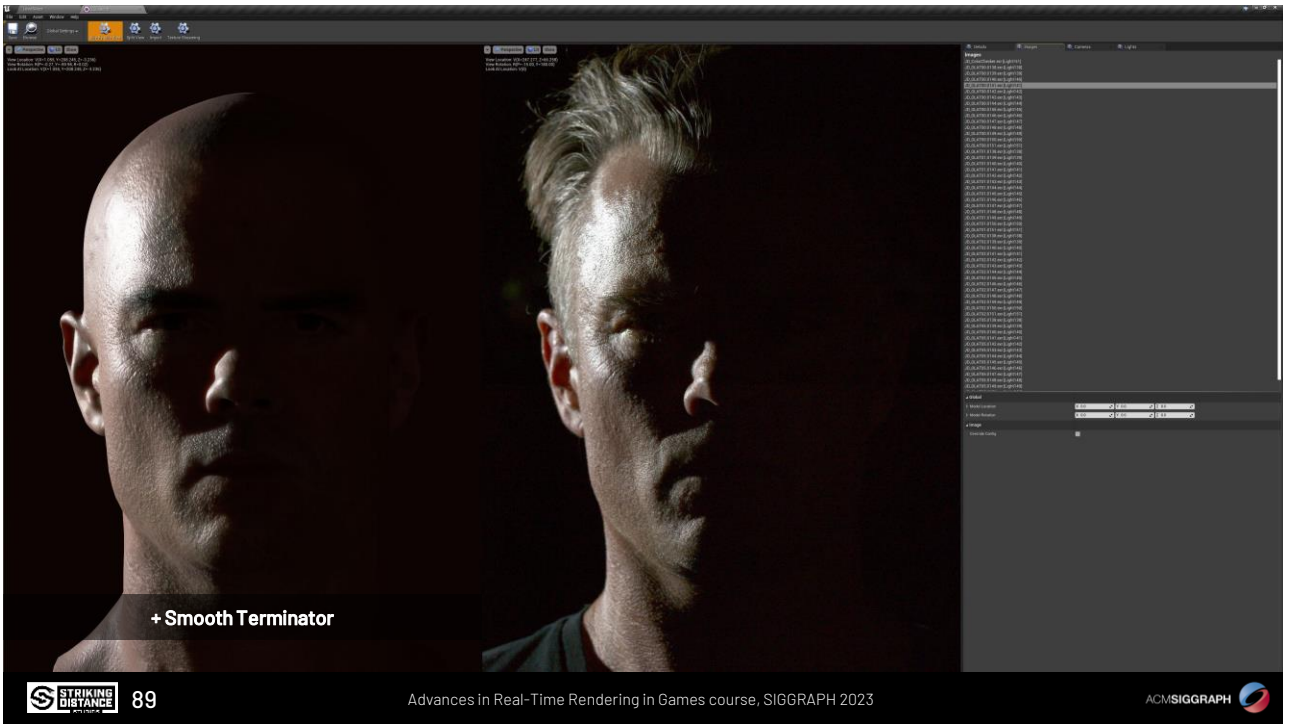
To get to that look, we added controls to smooth the terminators, which give us the results we see in this slide.

The traditional BRDF slice display does not show the impact of this feature effectively, so it is instead more convenient to look into the render directly.



It allowed for a much better match of our digital doubles.

Here we can see without a smooth terminator...



... and here with it.

[back and forth]

CALLISTO BRDF: SMOOTH TERMINATOR

LAMBERT + GGX BRDF

$$L_o = \left(f_{\text{lambert}}(\omega_i, \omega_o) + f_{\text{ggx}}(\omega_i, \omega_o) \right) L_i \overline{\cos \theta_i} d\omega_i$$

CALLISTO BRDF

$$L_o = \left(c_1(\theta_d, \theta_h) f_{\text{lambert}}(\omega_i, \omega_o) + f_{\text{ggx}}(\omega_i, \omega_o) \right) c_2(\theta_i) L_i \overline{\cos \theta_i} d\omega_i$$

$$c_2(\theta_i) = \underbrace{l(1, s(0, \alpha_s p, \cos \theta_i), \alpha_s o)}_{\text{Smooth Terminator}}$$

Smooth Terminator

$$\alpha_s = (1 - (1 - \theta_d)^3) (1 - (1 - \theta_h)^3)$$

$$l = \text{lerp} \\ s = \text{smoothstep}$$

| | Range | Default |
|------------------------------|-----------|---------|
| o Smooth Terminator | $[-1, 1]$ | 0 |
| p Smooth Terminator Length | $[0, 1]$ | 0.5 |

The implementation is straightforward, so I defer again to the offline slides for more details.

The mask defined by alpha allows for the smooth terminator to leave the retroreflection and Fresnel areas untouched, to make the parameters orthogonal, allowing to avoid iteration loops while setting the parameters.

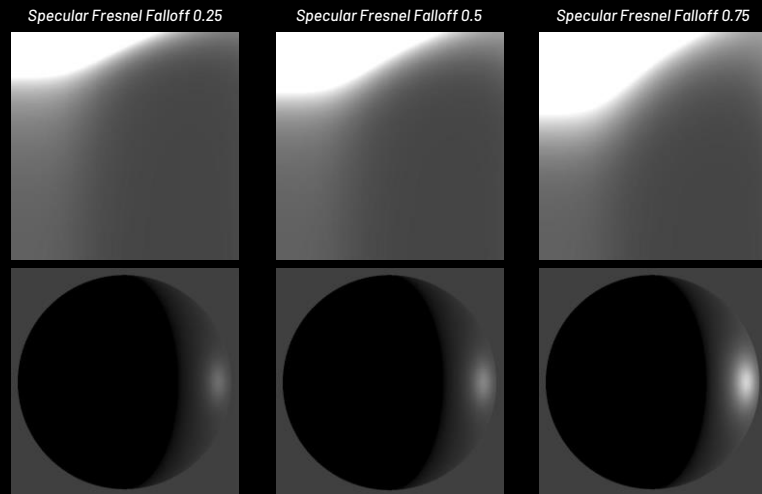


We love to backlit portrait shots.

One problem with them is that standard rendering techniques lack controls over the specular Fresnel peak.

And remember, what we have just presented was related to the diffuse response not speculars.

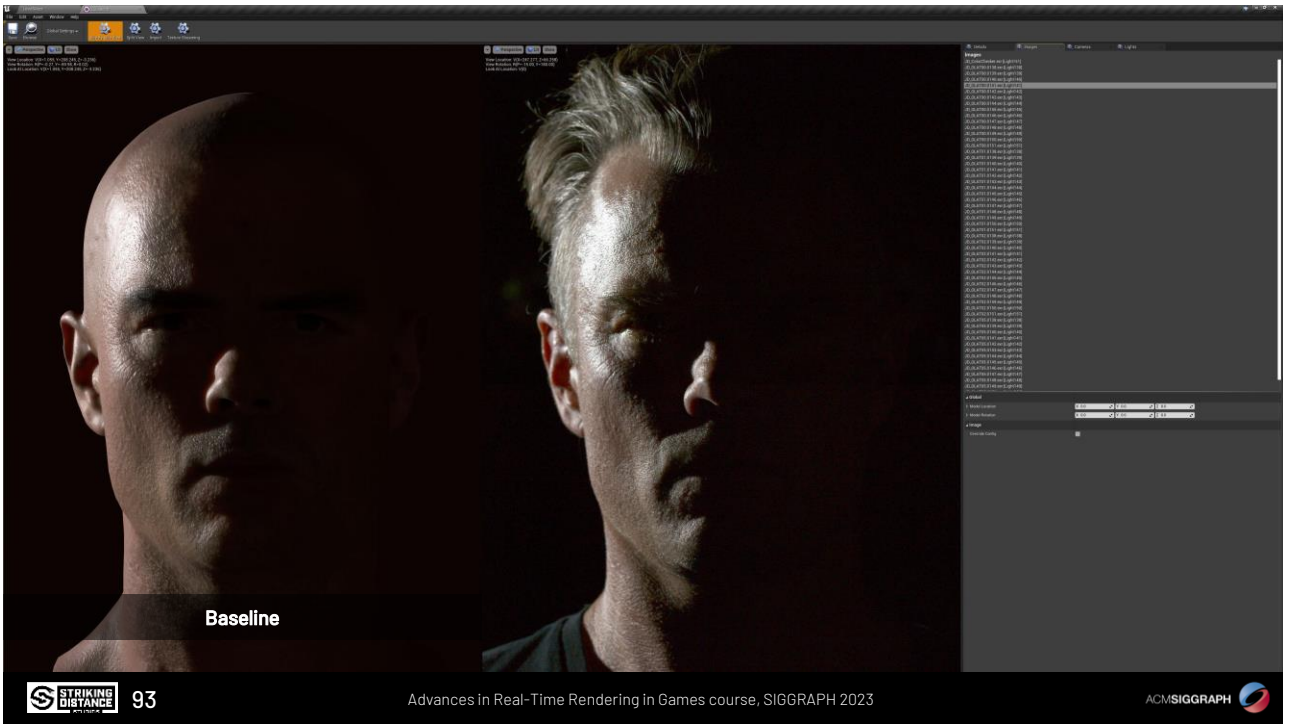
SPECULAR FRESNEL FALLOFF



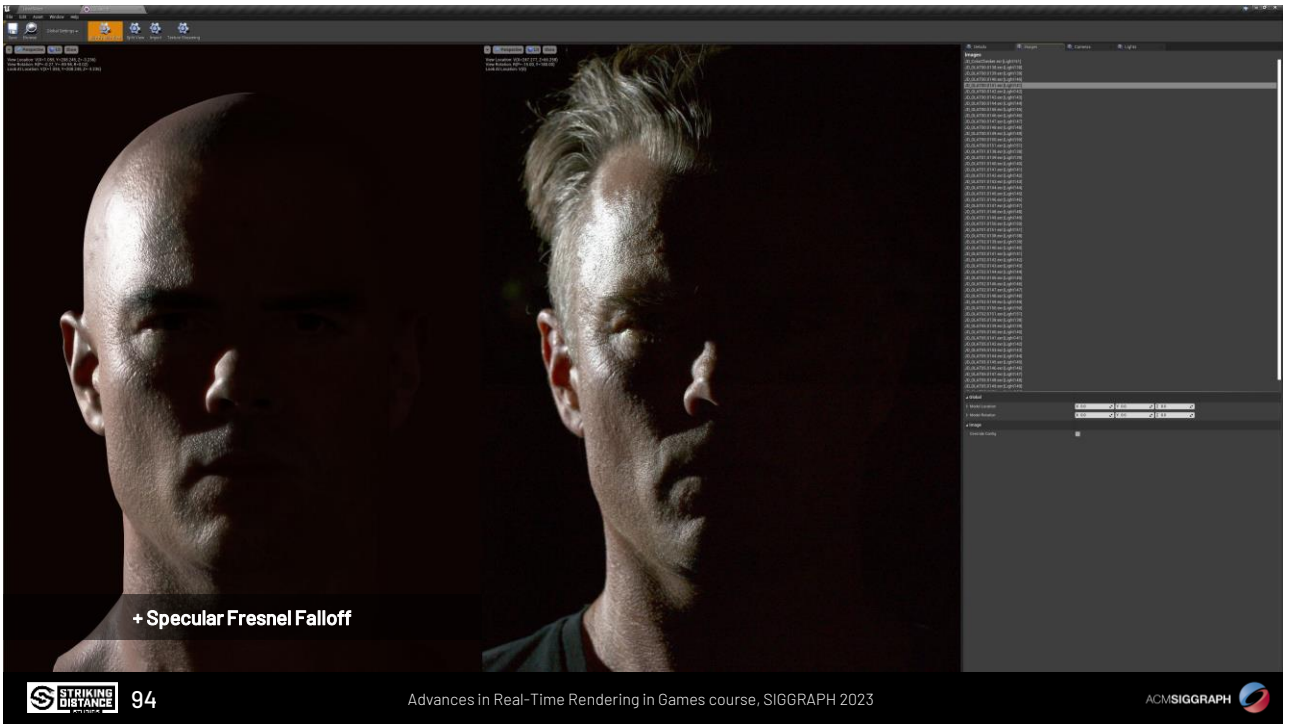
We verified that the Hoffman F82 parameter gave us the Fresnel control that we needed, but we wanted to evaluate if simpler solutions would be sufficient.

In that context, a simple change of the exponent in Schlick gave us the control we needed.

Rather than hardcoding to 5, we expose it as a parameter to control the behavior.



Applied to skin rendering, this is what we get with standard Fresnel controls...



...and this with the new ones.

CALLISTO BRDF: SPECULAR FRESNEL FALLOFF

LAMBERT + GGX BRDF

$$L_o = \left(f_{\text{lambert}}(\omega_i, \omega_o) + f_{\text{ggx}}(\omega_i, \omega_o) \right) L_i \overline{\cos \theta_i} d\omega_i$$

CALLISTO BRDF

$$L_o = \left(c_1(\theta_d, \theta_h) f_{\text{lambert}}(\omega_i, \omega_o) + f_{\text{ggx}}(\omega_i, \omega_o) \right) c_2(\theta_i) L_i \overline{\cos \theta_i} d\omega_i$$

$$f_{\text{ggx}}(\omega_i, \omega_o) = \frac{F G D}{4 \cos \theta_i \cos \theta_v}$$

$$F = f_0 + t(2 - r(n_s)) (1 - f_0)(1 - \cos \theta)^5 r(n_s)$$

$$r(x) = 2(1 - x)$$
$$t(x) = \min(\max(x, 0), 1)$$

| | Range | Default |
|------------------------------|--------|---------|
| n_s SpecularFresnelFalloff | [0, 1] | 0.5 |

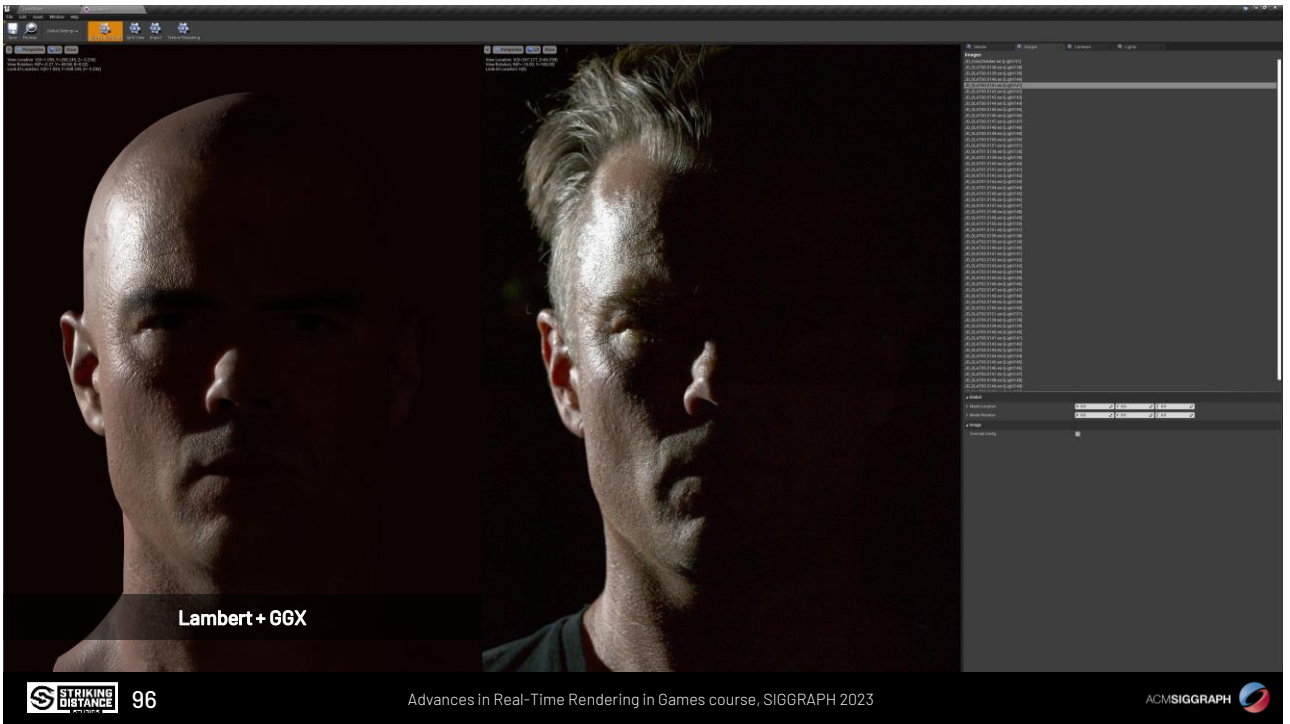
The implementation has two features.

The first one allows to control the power of Schlick's Fresnel.

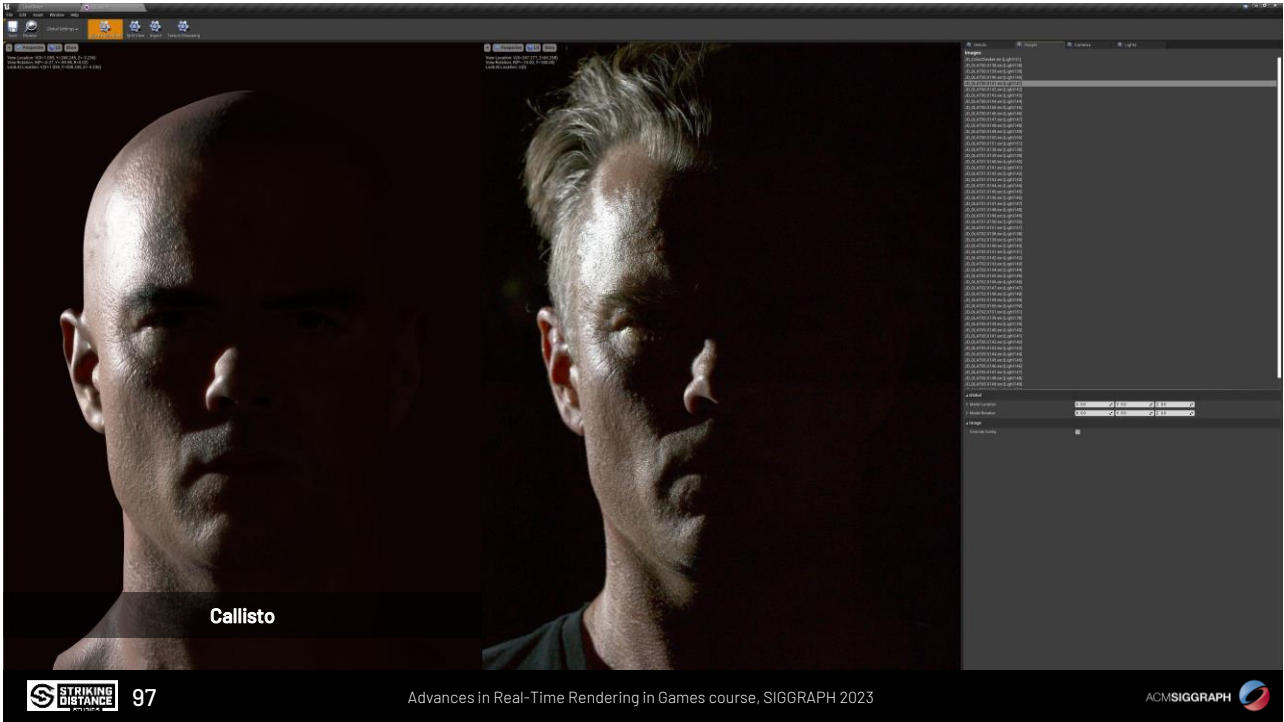
The second allows to fully remove the Fresnel component completely, which proven useful for matching some materials in the MERL database.

The remap $r(x)$ sets the parameter in 0 to 1 range, with 0.5 being the default value, which will make it match Schlick's Fresnel results.

We also used this parameterization for the diffuse falloffs presented before.

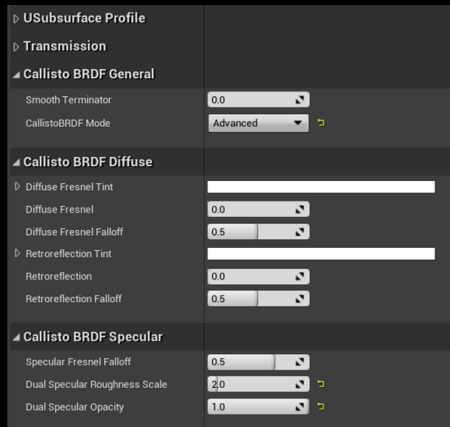


The combined effect of all Callisto terms can be strongly apparent in glazing angles, as we can see on this comparison...



[back and forth]

CALLISTO BRDF PARAMETERS



- Diffuse Fresnel
- Retroreflection
- Diffuse Fresnel Falloff
- Retroreflection Falloff
- Smooth Terminator

Base
Non-Specialized Users

- Diffuse Fresnel Tint
- Retroreflection Tint
- Smooth Terminator Tint
- Specular Fresnel Falloff
- Dual Specular Roughness Scale
- Dual Specular Opacity

Advanced
Technical / Material Artists

- Diffuse Fresnel Tangent Falloff
- Retroreflection Tangent Falloff
- Smooth Terminator Length

Full
Machine Learning

With the goal of being universal, we designed three tiers.

The first one, which we called Base, exposes five controls. The intensity of the diffuse Fresnel and retroreflection, the falloff at which they peak, and how smooth should be the terminators.


The second tier, called advanced, allows to specify the color of those components, the specular Fresnel falloff, and dual specular properties.

While the default values work well in most cases, the full model exposes all the parameters, avoiding all the constants and directly exposing them, which we thought useful for machine learning approaches.



Now how we use this in practice?

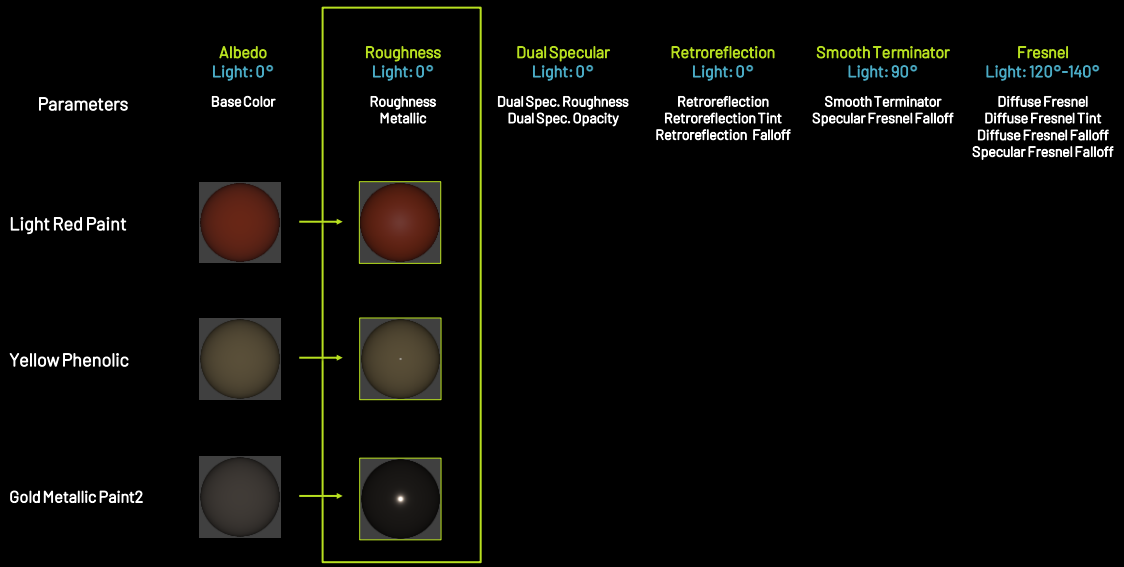
CALLISTO BRDF WORKFLOW

| Parameters | Albedo Light: 0° Base Color | Roughness Light: 0° Roughness Metallic | Dual Specular Light: 0° Dual Spec. Roughness Dual Spec. Opacity | Retroreflection Light: 0° Retroreflection Retroreflection Tint Retroreflection Falloff | Smooth Terminator Light: 90° Smooth Terminator Specular Fresnel Falloff | Fresnel Light: 120°-140° Diffuse Fresnel Diffuse Fresnel Tint Diffuse Fresnel Falloff Specular Fresnel Falloff |
|----------------------|---|---|--|--|--|---|
| Light Red Paint |  | | | | | |
| Yellow Phenolic |  | | | | | |
| Gold Metallic Paint2 |  | | | | | |

We first set the most basic parameter.

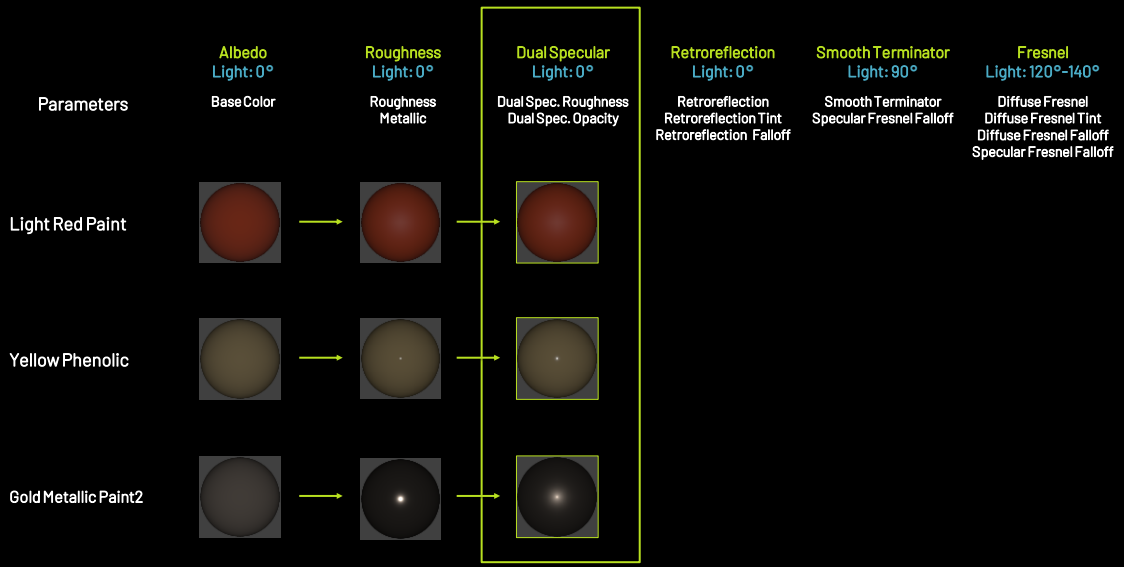
The base color.

CALLISTO BRDF WORKFLOW



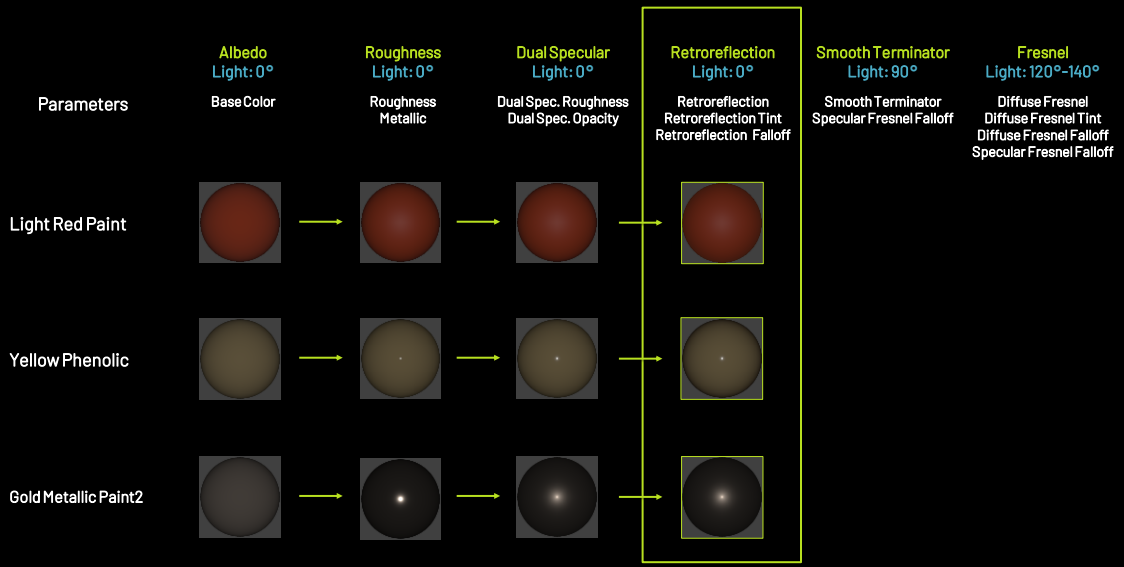
Next comes the roughness and metallic properties.

CALLISTO BRDF WORKFLOW



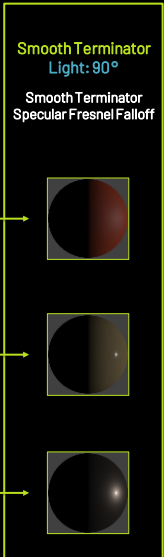
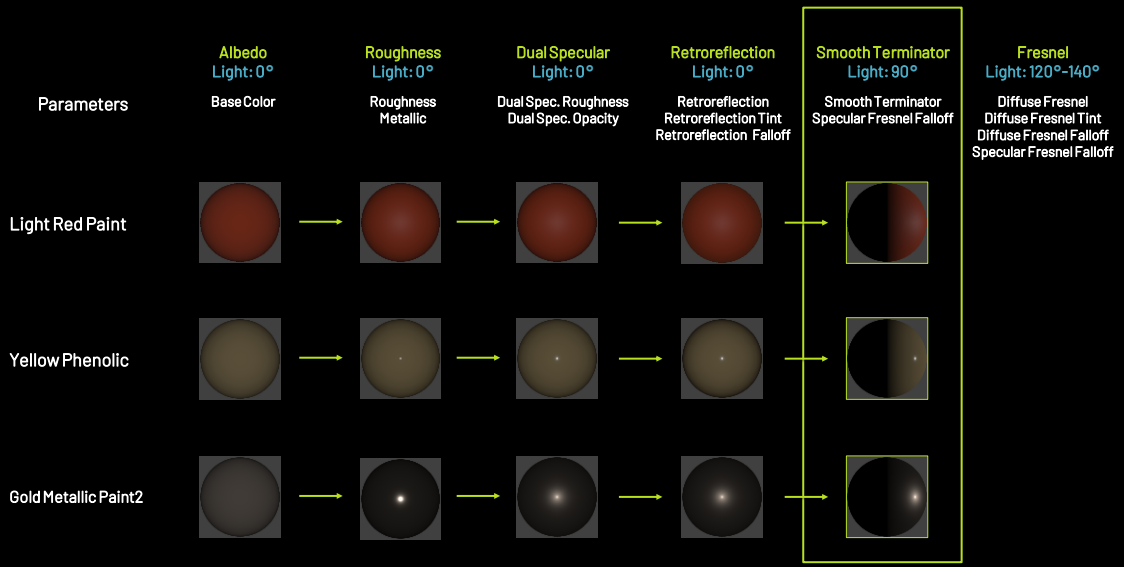
If the speculars cannot be matched with a single lobe, we adjust the dual specular properties.

CALLISTO BRDF WORKFLOW



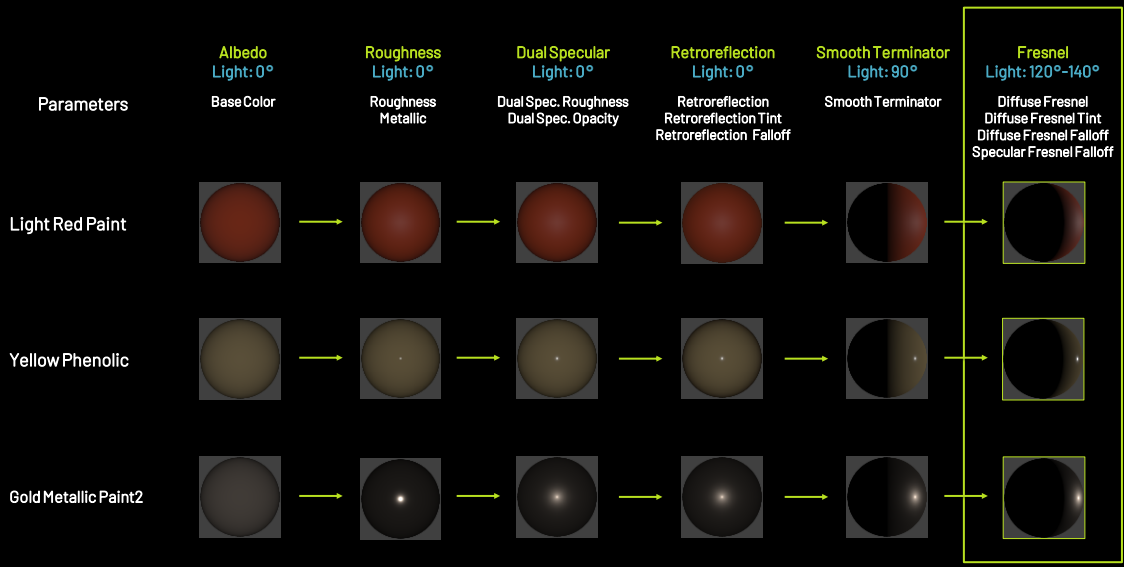
Then we set the diffuse retroreflection, flattening or enhancing the shape as the shading turns around the object.

CALLISTO BRDF WORKFLOW



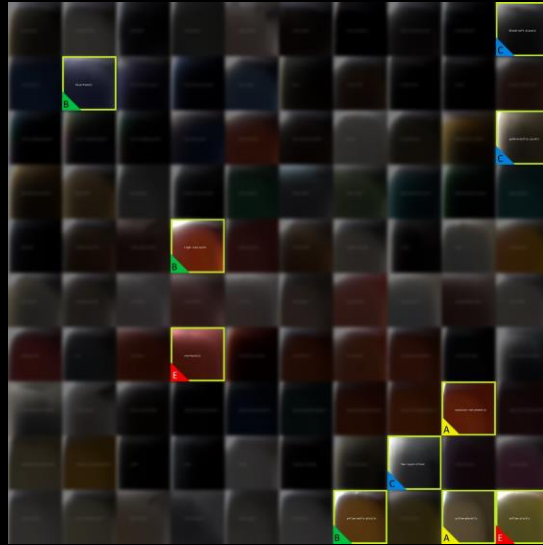
We then turn the light to the side, and smooth the shading terminators as necessary.

CALLISTO BRDF WORKFLOW



Finally, we turn the light further, and adjust the Fresnel properties, both for the diffuse and specular response.

CALLISTO BRDF SLICE SELECTION



The next question is, how this new model does against the MERL database?

To answer that, we selected 2 materials per type, and fitted them with both Burley and Callisto.

CALLISTO BRDF SLICE SELECTION

Black Soft Plastic

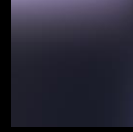
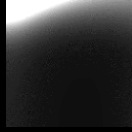
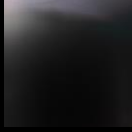
Burley

Callisto

Blue Fabric

Burley

Callisto



Error 14.75%

Error 3.84%

Error 11.63%

Error 2.79%

- Base Color: (20, 20, 20)
- Metallic: 0
- Subsurface: 0
- Specular: 0.2
- Roughness: 0.6
- Specular Tint: 0
- Anisotropic: 0
- Sheen: 0
- Sheen Tint: 0.5
- Clear Coat: 0
- Clear Coat Gloss: 1

- Base Color: (20, 20, 20)
- Diffuse Fresnel Tint: (255, 255, 255)
- Diffuse Fresnel: 0.75
- Diffuse Fresnel Falloff: 0.7
- Diffuse Fresnel Tangent Falloff: 0.75
- Retroreflection Tint: (255, 255, 255)
- Retroreflection: 3
- Retroreflection Falloff: 0.7
- Retroreflection Tangent Falloff: 0.75
- Smooth Terminator: 0
- Metallic: 0
- Specular: 0.2
- Specular Fresnel Falloff: 0
- Roughness: 0.6
- Dual Specular Roughness Scale: 2
- Dual Specular Opacity: 0

- Base Color: (38, 44, 66)
- Metallic: 0
- Subsurface: 0
- Specular: 0.25
- Roughness: 0.6
- Specular Tint: 0
- Anisotropic: 0
- Sheen: 0.15
- Sheen Tint: 1
- Clear Coat: 0
- Clear Coat Gloss: 1

- Base Color: (40, 42, 66)
- Diffuse Fresnel Tint: (255, 218, 185)
- Diffuse Fresnel: 4.0
- Diffuse Fresnel Falloff: 0.6
- Diffuse Fresnel Tangent Falloff: 0.95
- Retroreflection Tint: (255, 255, 255)
- Retroreflection: 3
- Retroreflection Falloff: 0.5
- Retroreflection Tangent Falloff: 0.75
- Smooth Terminator: 0
- Metallic: 0
- Specular: 0.2
- Specular Fresnel Falloff: 0
- Roughness: 0.77
- Dual Specular Roughness Scale: 2
- Dual Specular Opacity: 0

Given the explicit controls, as expected, we can see it matches well the captured data.

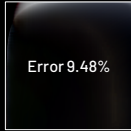
For time constrains, we are leaving the remaining comparisons for the offline slides.

CALLISTO BRDF SLICE SELECTION

Yellow Phenolic

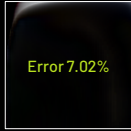


Burley



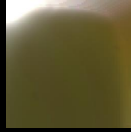
- Base Color: (148, 125, 92)
- Metallic: 0
- Subsurface: 0
- Specular: 0.5
- Roughness: 0.04
- Specular Tint: 0
- Anisotropic: 0
- Sheen: 0
- Sheen Tint: 0.5
- Clear Coat: 1
- Clear Coat Gloss: 0.9

Callisto

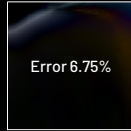
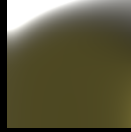


- Base Color: (152, 130, 93)
- Diffuse Fresnel Tint: (255, 255, 255)
- Diffuse Fresnel: 0.8
- Diffuse Fresnel Falloff: 0.85
- Diffuse Fresnel Tangent Falloff: 1
- Retroreflection Tint: (255, 255, 255)
- Retroreflection: 0.3
- Retroreflection Falloff: 0.85
- Retroreflection Tangent Falloff: 1
- Smooth Terminator: 0.25
- Metallic: 0
- Specular: 0.5
- Specular Fresnel Falloff: 0.5
- Roughness: 0.055
- Dual Specular Roughness Scale: 4.25
- Dual Specular Opacity: 0.15

Yellow Plastic

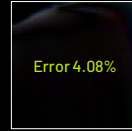
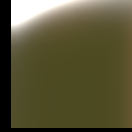


Burley



- Base Color: (130, 121, 56)
- Metallic: 0
- Subsurface: 0
- Specular: 0.05
- Roughness: 0.5
- Specular Tint: 0
- Anisotropic: 0
- Sheen: 0
- Sheen Tint: 0.5
- Clear Coat: 0
- Clear Coat Gloss: 1

Callisto



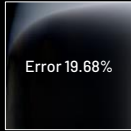
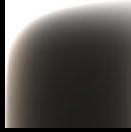
- Base Color: (128, 124, 55)
- Diffuse Fresnel Tint: (255, 200, 255)
- Diffuse Fresnel: 5
- Diffuse Fresnel Falloff: 0.6
- Diffuse Fresnel Tangent Falloff: 0.85
- Retroreflection Tint: (225, 191, 255)
- Retroreflection: 3.5
- Retroreflection Falloff: 0.25
- Retroreflection Tangent Falloff: 1
- Smooth Terminator: 0
- Metallic: 0
- Specular: 0.1
- Specular Fresnel Falloff: 0.375
- Roughness: 0.5
- Dual Specular Roughness Scale: 2
- Dual Specular Opacity: 0

CALLISTO BRDF SLICE SELECTION

Gold Metallic Paint 2

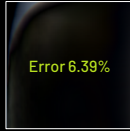
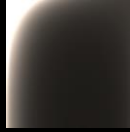


Burley



- Base Color: (105, 95, 84)
- Metallic: 0.86
- Subsurface: 0
- Specular: 1
- Roughness: 0.45
- Specular Tint: 0.889
- Anisotropic: 0
- Sheen: 0.77
- Sheen Tint: 0.5
- Clear Coat: 3
- Clear Coat Gloss: 0.75

Callisto

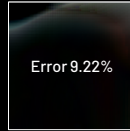
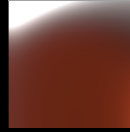


- Base Color: (105, 86, 88)
- Diffuse Fresnel Tint: (255, 255, 255)
- Diffuse Fresnel: 1
- Diffuse Fresnel Falloff: 0.5
- Diffuse Fresnel Tangent Falloff: 1
- Retroreflection Tint: (255, 255, 255)
- Retroreflection: 1
- Retroreflection Falloff: 0.5
- Retroreflection Tangent Falloff: 0.75
- Smooth Terminator: 0
- Metallic: 0.85
- Specular: 1
- Specular Fresnel Falloff: 0
- Roughness: 0.21
- Dual Specular Roughness Scale: 2
- Dual Specular Opacity: 0.9

Light Red Paint

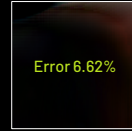
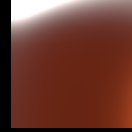


Burley



- Base Color: (173, 60, 27)
- Metallic: 0
- Subsurface: 0
- Specular: 0.175
- Roughness: 0.55
- Specular Tint: 0
- Anisotropic: 0
- Sheen: 0
- Sheen Tint: 0.5
- Clear Coat: 0
- Clear Coat Gloss: 1

Callisto



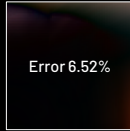
- Base Color: (173, 58, 23)
- Diffuse Fresnel Tint: (255, 255, 255)
- Diffuse Fresnel: 1
- Diffuse Fresnel Falloff: 0.5
- Diffuse Fresnel Tangent Falloff: 1
- Retroreflection Tint: (200, 240, 255)
- Retroreflection: 3.5
- Retroreflection Falloff: 0.5
- Retroreflection Tangent Falloff: 0.65
- Smooth Terminator: 0.2
- Metallic: 0
- Specular: 0.25
- Specular Fresnel Falloff: 0.4
- Roughness: 0.8
- Dual Specular Roughness Scale: 2
- Dual Specular Opacity: 0

CALLISTO BRDF SLICE SELECTION

Red Fabric 2

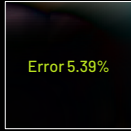


Burley



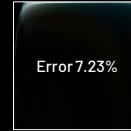
- Base Color: (105, 33, 10)
- Metallic: 0
- Subsurface: 0
- Specular: 0.5
- Roughness: 1
- Specular Tint: 0
- Anisotropic: 0
- Sheen: 0.2
- Sheen Tint: 1
- Clear Coat: 0
- Clear Coat Gloss: 1

Callisto



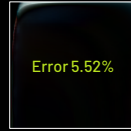
- Base Color: (106, 29, 8)
- Diffuse Fresnel Tint: (180, 160, 255)
- Diffuse Fresnel: 25
- Diffuse Fresnel Falloff: 0.6
- Diffuse Fresnel Tangent Falloff: 0.97
- Retreflection Tint: (255, 0, 255)
- Retreflection: 1.25
- Retreflection Falloff: 0.5
- Retreflection Tangent Falloff: 0.75
- Smooth Terminator: -0.1
- Metallic: 0
- Specular: 0.5
- Specular Fresnel Falloff: 0.5
- Roughness: 1
- Dual Specular Roughness Scale: 2
- Dual Specular Opacity: 0

Specular Red Phenolic



- Base Color: (151, 58, 27)
- Metallic: 0
- Subsurface: 0
- Specular: 0.5
- Roughness: 0.057
- Specular Tint: 0
- Anisotropic: 0
- Sheen: 0
- Sheen Tint: 0.5
- Clear Coat: 0.35
- Clear Coat Gloss: 1

Callisto



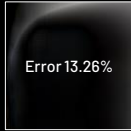
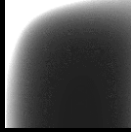
- Base Color: (154, 57, 32)
- Diffuse Fresnel Tint: (255, 255, 255)
- Diffuse Fresnel: 0.6
- Diffuse Fresnel Falloff: 0.9
- Diffuse Fresnel Tangent Falloff: 1
- Retreflection Tint: (255, 255, 255)
- Retreflection: 0.5
- Retreflection Falloff: 0.8
- Retreflection Tangent Falloff: 0.95
- Smooth Terminator: 0.2
- Metallic: 0
- Specular: 0.55
- Specular Fresnel Falloff: 0.5
- Roughness: 0.03
- Dual Specular Roughness Scale: 4
- Dual Specular Opacity: 0.125

CALLISTO BRDF SLICE SELECTION

Two Layer Silver

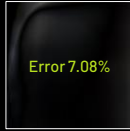
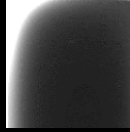


Burley



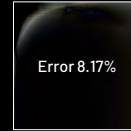
- Base Color: (125, 125, 125)
- Metallic: 0.9
- Subsurface: 0
- Specular: 0.5
- Roughness: 0.41
- Specular Tint: 1
- Anisotropic: 0
- Sheen: 0.5
- Sheen Tint: 1
- Clear Coat: 1
- Clear Coat Gloss: 0.9

Callisto



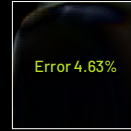
- Base Color: (250, 250, 250)
- Diffuse Fresnel Tint: (255, 255, 255)
- Diffuse Fresnel: 1
- Diffuse Fresnel Falloff: 0.5
- Diffuse Fresnel Tangent Falloff: 1
- Retroreflection Tint: (255, 255, 255)
- Retroreflection: 1
- Retroreflection Falloff: 0.5
- Retroreflection Tangent Falloff: 0.75
- Smooth Terminator: 0
- Metallic: 0.37
- Specular: 0.5
- Specular Fresnel Falloff: 0.5
- Roughness: 0.027
- Dual Specular Roughness Scale: 14
- Dual Specular Opacity: 0.18

Yellow Matte Plastic



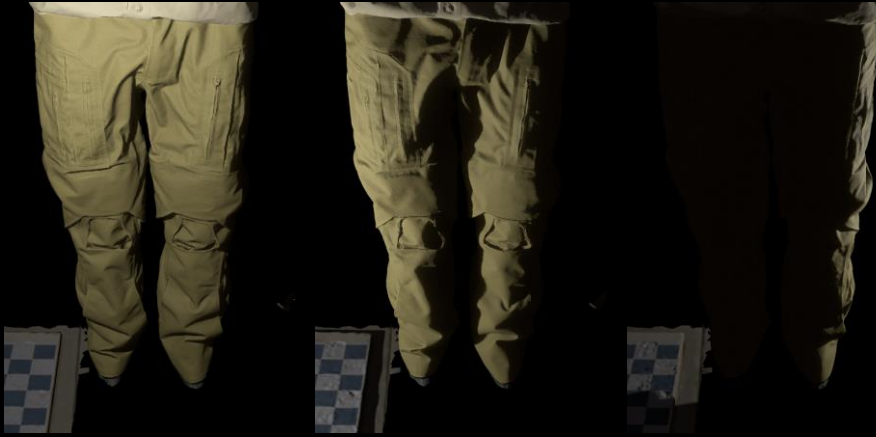
- Base Color: (145, 86, 43)
- Metallic: 0
- Subsurface: 0
- Specular: 0.15
- Roughness: 0.3
- Specular Tint: 0
- Anisotropic: 0
- Sheen: 0
- Sheen Tint: 0.5
- Clear Coat: 0
- Clear Coat Gloss: 1

Callisto



- Base Color: (148, 97, 48)
- Diffuse Fresnel Tint: (255, 255, 255)
- Diffuse Fresnel: 0.25
- Diffuse Fresnel Falloff: 0.75
- Diffuse Fresnel Tangent Falloff: 1
- Retroreflection Tint: (255, 255, 255)
- Retroreflection: 0.5
- Retroreflection Falloff: 0.85
- Retroreflection Tangent Falloff: 1
- Smooth Terminator: 0.25
- Metallic: 0
- Specular: 0.16
- Specular Fresnel Falloff: 0.25
- Roughness: 0.165
- Dual Specular Roughness Scale: 2
- Dual Specular Opacity: 0.9

CALLISTO BRDF ON THE PRISONER SUIT



Lambert + GGX

We are going to show the results of Callisto BRDF on the prisoner suit.

This is Lambert and GGX...

CALLISTO BRDF ON THE PRISONER SUIT



Photo Reference

...this is the photo reference...

CALLISTO BRDF ON THE PRISONER SUIT



Callisto

...and this Callisto BRDF.

We found Callisto BRDF to better capture the characteristics of the cloth we used as reference for the prisoner suit.

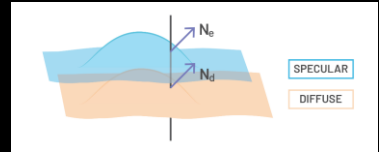
DUAL NORMAL



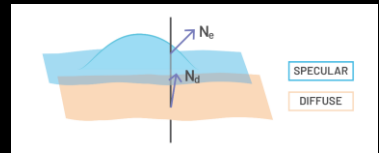
Standard



Dual Normal



Standard



Dual Normal

We added support for dual normals, split for specular and diffuse.

This was important to render sweat in a more accurate way.

Material consistency over distance.

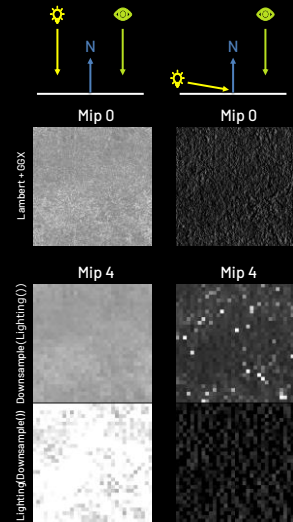
We have described so far how we work on consistency over light and view direction.

But how we can make materials more consistent over distance?

[Note that the ideas in this section did not shipped in our game, as they were developed late in the product cycle]

PROBLEM: LIGHTING AND MIPMAPPING

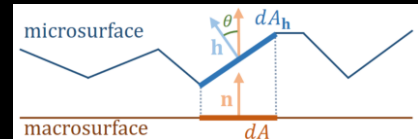
- $\text{Lighting}(\text{Downsample}()) \neq \text{Downsample}(\text{Lighting}())$
- As surfaces get further away, information is lost
 - Resulting in incorrect lighting
- Caused by mipmapping Normal and Roughness input maps
- Texture Space Lighting? (GT)
 - We can't pay for this



For that we can start describing the problem space. It is not the same to downsampling the inputs first, then performing the lighting, than lighting first, then performing the downsampling, which would be the same as using texture space lighting, which we will use as our ground truth.

SPECULAR ANTIALIASING

- [Chan18] Material Advances in Call of Duty: WWII
 - Two versions: Macro vs Micro surface information
- Offline Preprocess of normal maps
- Establish a relationship between normal length and roughness
 - Convert lost normal length to roughness to use at runtime
- We have compared with ground truth to see how it compares for the specular



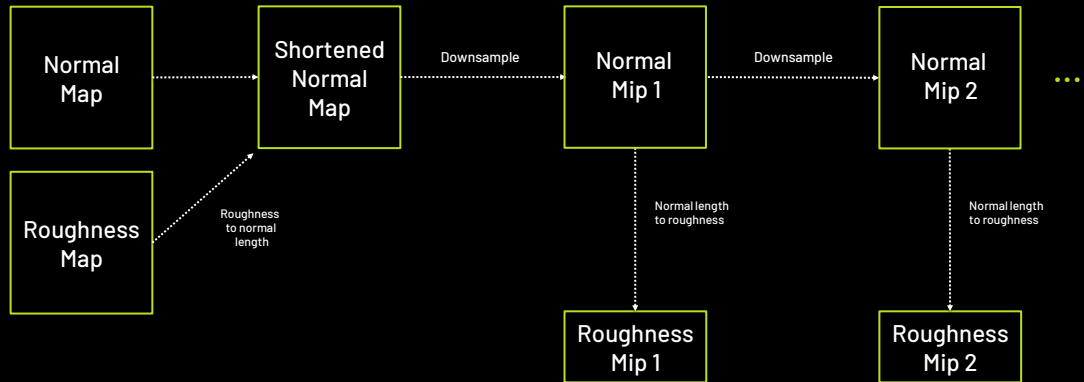
To mitigate this problem [Chan18] established a relation from normal lengths and roughness.

The premise is that a normal from a normal map eventually becomes a microfacet when seen from far away.

This allows to take roughness information created by normal maps and store them into the roughness maps.

SPECULAR ANTIALIASING [CHAN18]

Offline

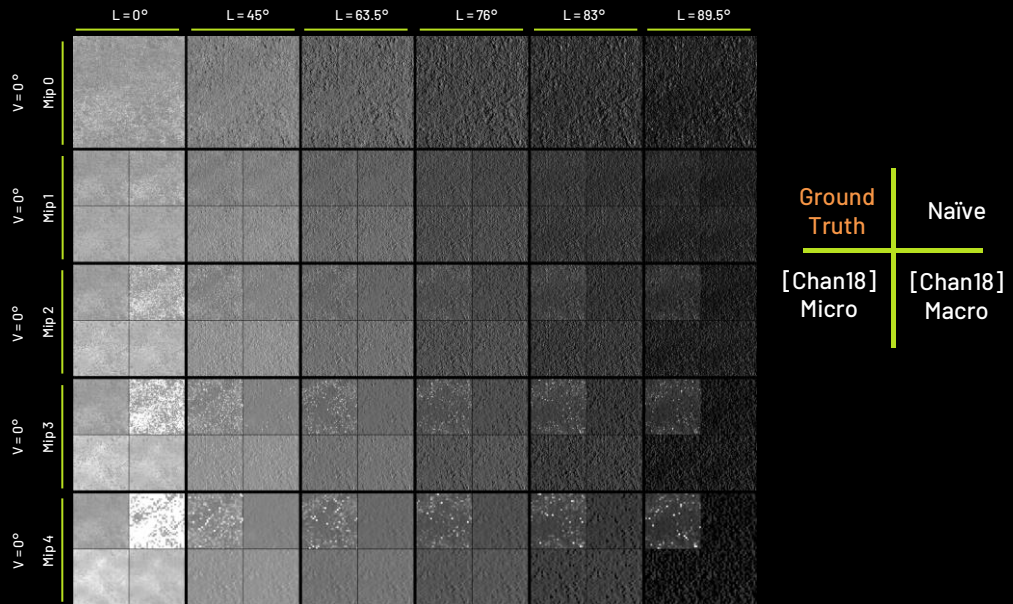


The process is as follows.

Roughness is converted to normal length, and then combined with the normal map.

These normal maps are downsampled for mipmapping, and then the resulting normal lengths are converted back to roughness values.

SPECULAR ANTIALIASING [CHAN18] RESULTS

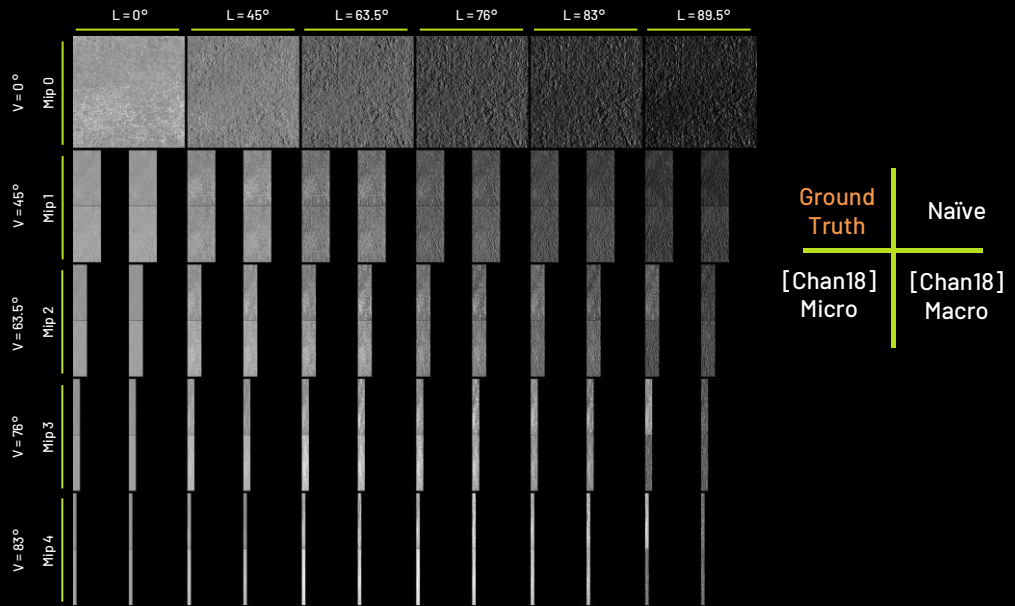


Here we see how [Chan18] micro and macro versions compare with the ground truth, and the improvements over doing nothing about the problem, which we called *naïve*.

There is a drastic improvement in the speculars for some of the cases.

[Note that we used 50% reflectance at normal incidence to help visualization]

SPECULAR ANTIALIASING [CHAN18] RESULTS



And here we have results for different view angles.

DIFFUSE ANTIALIASING

- Can we apply [Chan18] to Diffuse?

- Yes, but we would need to introduce microsurface information (roughness)

- [Heitz14] Understanding the Masking-Shadowing Function in Microfacet-Based BRDFs

[Heitz14]

- Defines this but no closed form solution is provided

$$\rho(\omega_o, \omega_i) = \frac{1}{\pi} \frac{1}{|\omega_g \omega_o| |\omega_g \omega_i|} \int_{\Omega} \langle \omega_o, \omega_m \rangle \langle \omega_i, \omega_m \rangle G_2(\omega_o, \omega_i, \omega_m) D(\omega_m) d\omega_m$$

- We wanted to create a LUT

- But would have 4 dimensions

- Our initial approach:

Ours

- We fixed ϕ , assuming Light and View Coplanar
- Reduces the dimensions to 3 (3D LUT)
- Assume $G_2 = 1$ as normal maps do not have visibility information

$$\rho(\omega_o, \omega_i) = \frac{1}{\pi} \frac{1}{|\omega_g \omega_o| |\omega_g \omega_i|} \int_{\Omega} \langle \omega_o, \omega_m \rangle \langle \omega_i, \omega_m \rangle \cancel{G_2}(\omega_o, \omega_i, \omega_m) D(\omega_m) d\omega_m$$

$$\phi_v = \phi_l$$



122

(Developed with Jose Naranjo, Jon Diego, Jay Ryness, and Miguel Rodriguez)

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023



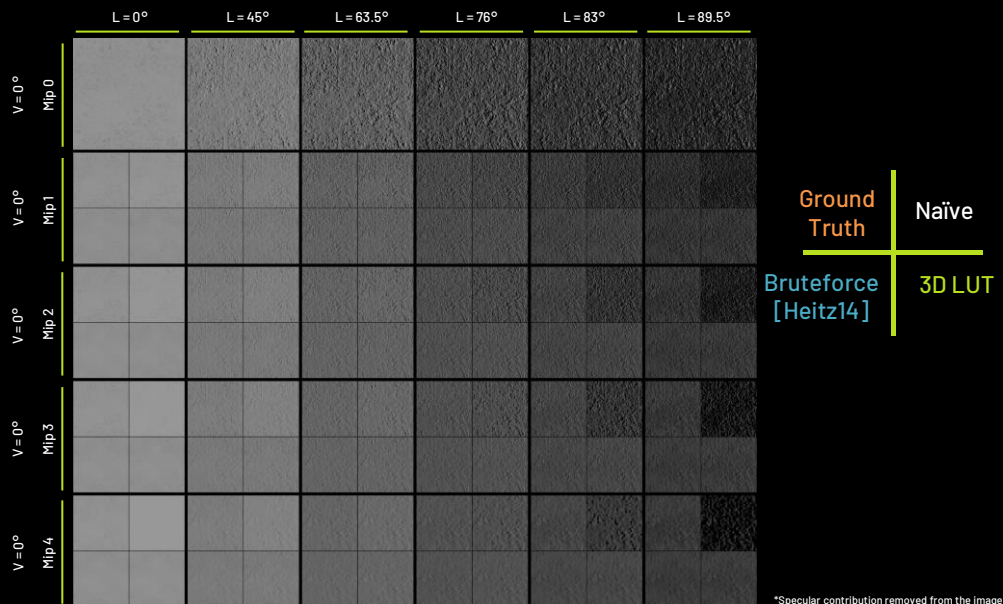
[Chan18] solution is only applied to the specular component.

We would like to also perform the same approximation for the diffuse. For that we would need a diffuse model that would account for roughness, and using the same microsurface as GGX, so a single map could be shared for both specular and diffuse.

[Heitz14] defines such a model, but unfortunately does not have a closed form solution.

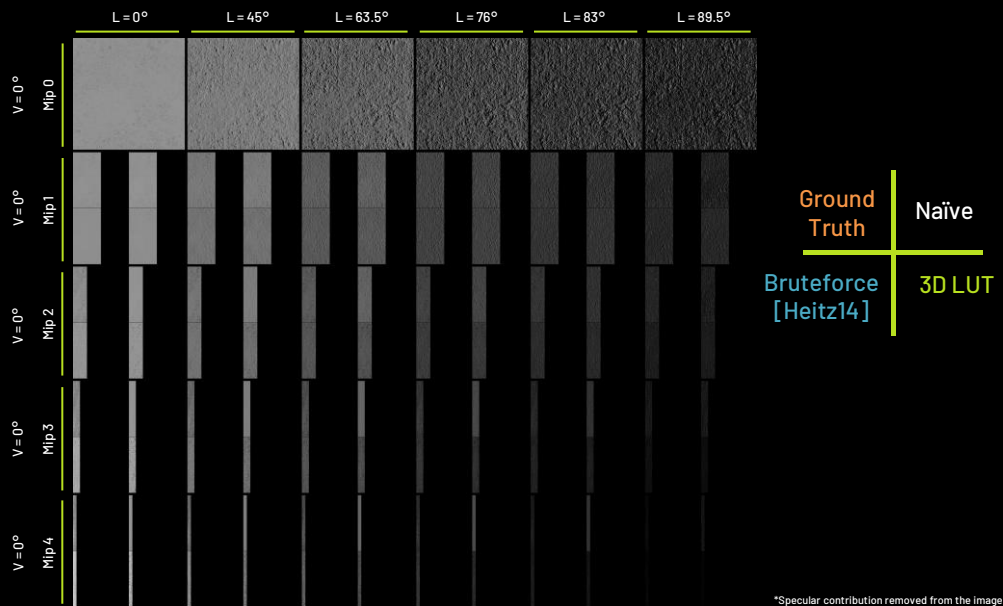
We have created a LUT for that function, assuming light and view are coplanar, which allows to reduce the number of dimensions to 3.

DIFFUSE ANTIALIASING: 3D LUT



Here are some results of the 3D LUT approximation, compared with the ground truth and bruteforce evaluating [Heitz2014] when viewing the surface from the front...

DIFFUSE ANTIALIASING: 3D LUT

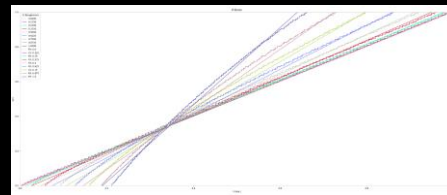
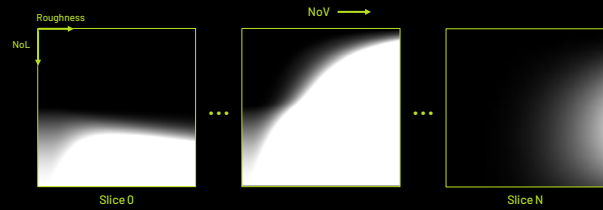


*Specular contribution removed from the image

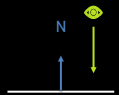
...and from grazing angles.

FUNCTIONAL APPROXIMATION: PROXIMA BRDF

- Observing the 3D LUT:
 - Most representative:
 - Normal and View are similar
 - Light and Normal close to 90 degrees
 - Similar to Lambert:
 - Roughness approaching zero
 - Other Light and Normal angles
- Functional approximation to the 3D LUT:
 - 2D fit for Normal and View being equal
 - Fallback to Lambert as Light and View go past 90 degrees
- We called this functional approximation:
 - Proxima BRDF



Slices of the LUT most representative section and line fits (9 roughness values)



Next step was making a functional approximation.

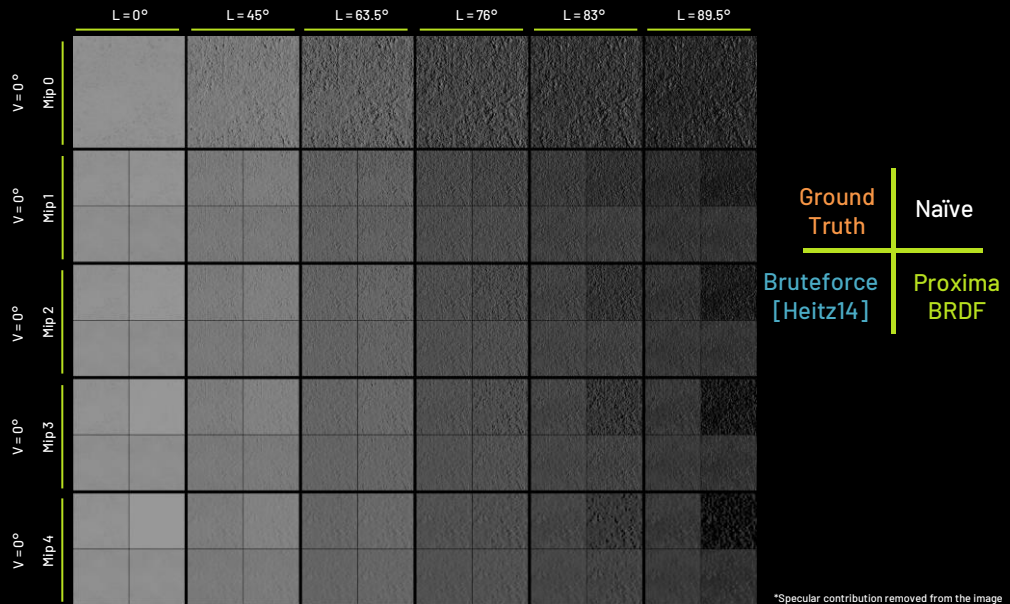
We observed the most characteristic behavior to happen when light and normal approximate 90 degrees from each other, and when normal and view are similar.

Other cases were well approximated by Lambert.

Following those observations, we first fitted multiple roughness cases for this most representative cases, and then we fallback to Lambert in other cases.

We called our functional approximation, Proxima BRDF.

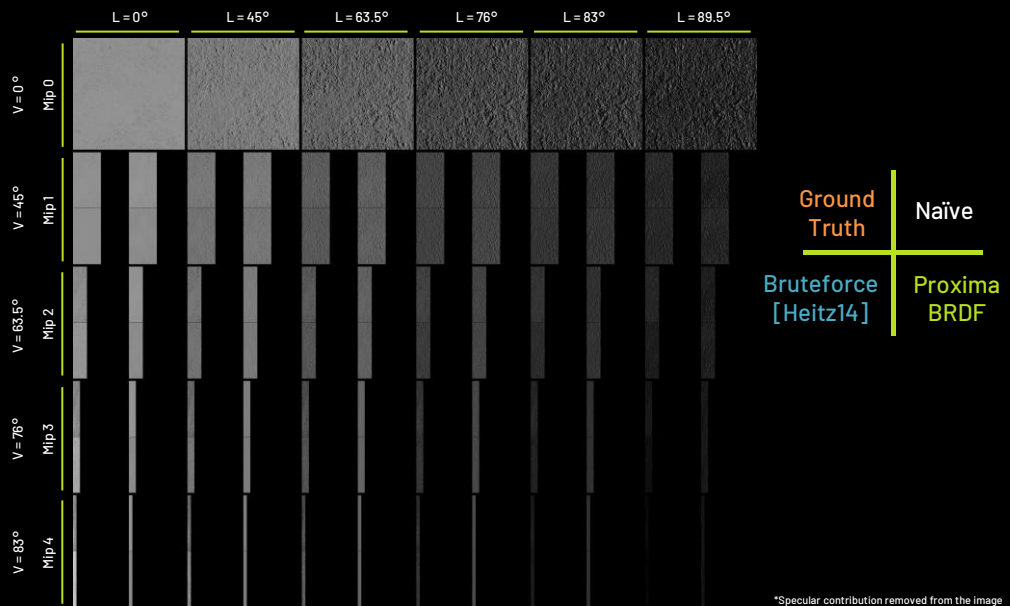
PROXIMA BRDF: RESULTS



*Specular contribution removed from the image

Here we can see the results of Proxima BRDF against the ground truth, and brute force evaluating [Heitz14] with [Chan18], for a frontal view.

PROXIMA BRDF: RESULTS



*Specular contribution removed from the image

And here for grazing angles.

PROXIMA BRDF

LAMBERT + GGX BRDF
$$L_o = \left(f_{\text{lambert}}(\omega_i, \omega_o) + f_{\text{ggx}}(\omega_i, \omega_o) \right) L_i \overline{\cos \theta_i} d\omega_i$$

PROXIMA BRDF + GGX BRDF
$$L_o = \left(f_{\text{proxima}}(\omega_i, \omega_o) + f_{\text{ggx}}(\omega_i, \omega_o) \right) L_i \overline{\cos \theta_i} d\omega_i$$

$$f_{\text{proxima}}(\omega_i, \omega_o) = \frac{\rho}{\pi} \left(\alpha (-0.55 + 0.19 \overline{\cos \theta_i}^{-1}) (1 - \overline{\cos \theta_k}^{1/2}) + 1 \right)$$

$$\cos \theta_k = -V \cdot L$$

| | |
|----------------------|--------------------|
| f_{proxima} | Proxima BRDF |
| f_{lambert} | $\frac{\rho}{\pi}$ |
| α | GGX Alpha |

Notes:

- α here has a different meaning to the α in other slices
- For numerical stability we recommend to pre-multiply by $\overline{\cos \theta_i}$ to remove the division

Finally, here we have the math for our functional approximation to the GGX-based microfacet diffuse BRDF, which we termed Proxima BRDF.

CALLISTO + PROXIMA BRDF: RESULTS



One question is, how we combine Callisto and Proxima BRDF together?

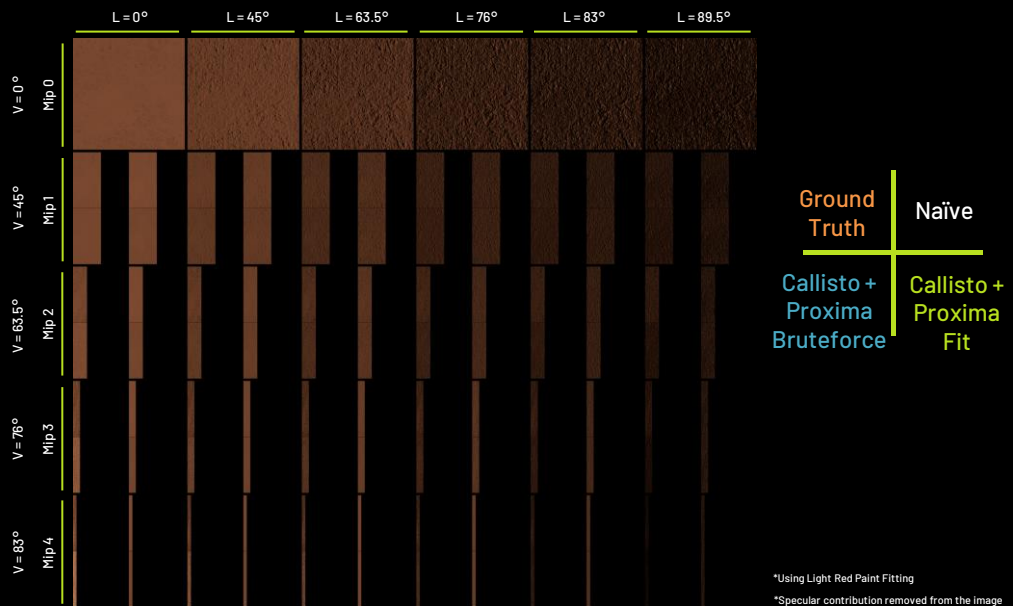
Given the number of parameters that Callisto has, it would not be practical to create a LUT for how it behaves over the distance.

Instead, we just approximated it by replacing the lambert term by Proxima (math will be shown later on).

Here we compare doing supersampling of Callisto together with Proxima (in blue), versus the approximation of replacing the Lambert term by Proxima, within the Callisto BRDF.

We did comparisons for a front view...

CALLISTO + PROXIMA BRDF: RESULTS



*Using Light Red Paint Fitting

*Specular contribution removed from the image



130

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023



...and for grazing angles as well.

We found the results to hold up well in practice when using this approximation.

CALLISTO BRDF + PROXIMA BRDF

$$\text{CALLISTO+GGX BRDF} \quad L_o = \left(c_1(\theta_d, \theta_h) f_{\text{lambert}}(\omega_i, \omega_o) + f_{\text{ggx}}(\omega_i, \omega_o) \right) c_2(\theta_i) L_i \overline{\cos \theta_i} d\omega_i$$

$$\text{CALLISTO+GGX+PROXIMA BRDF} \quad L_o = \left(c_1(\theta_d, \theta_h) f_{\text{proxima}}(\omega_i, \omega_o) + f_{\text{ggx}}(\omega_i, \omega_o) \right) c_2(\theta_i) L_i \overline{\cos \theta_i} d\omega_i$$

$$f_{\text{proxima}}(\omega_i, \omega_o) = \frac{\rho}{\pi} \left(\alpha (-0.55 + 0.19 \overline{\cos \theta_i}^{-1}) (1 - \overline{\cos \theta_k}^{1/2}) + 1 \right)$$

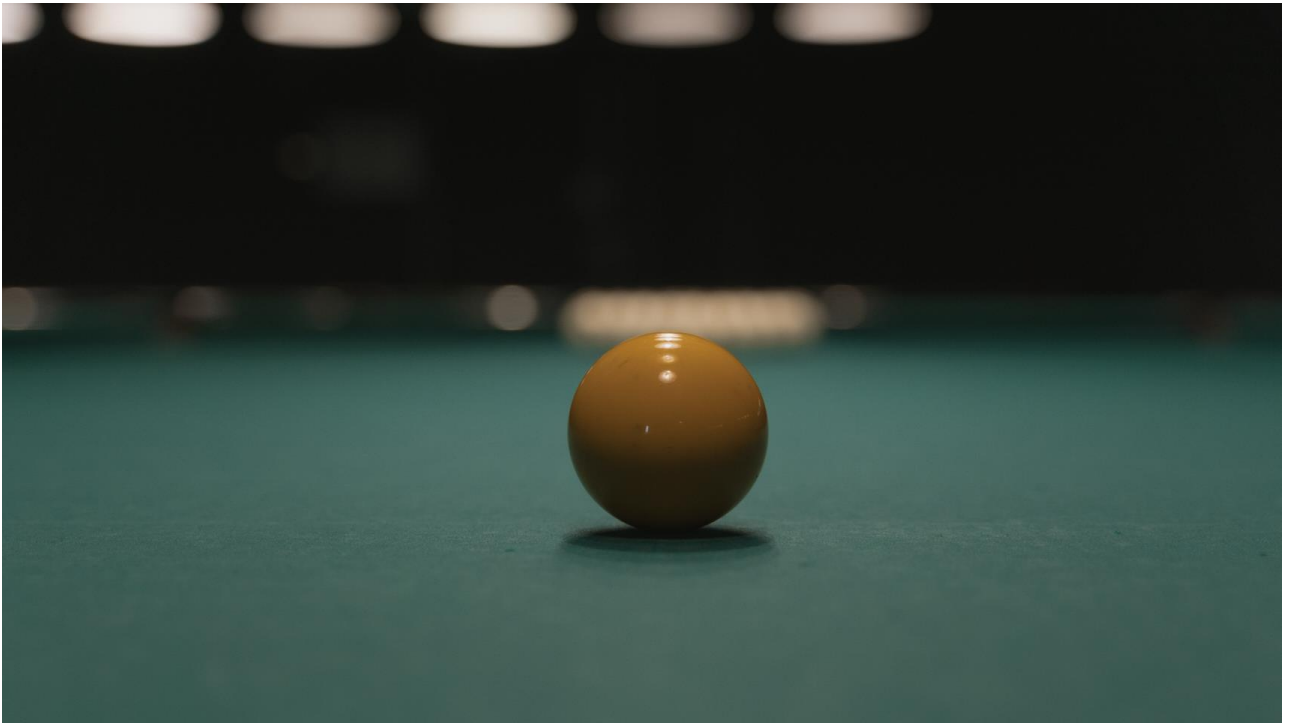
$$\cos \theta_k = -V \cdot L$$

| | |
|----------------------|--------------------|
| f_{proxima} | Proxima BRDF |
| f_{lambert} | $\frac{\rho}{\pi}$ |
| α | GGX Alpha |

Notes:

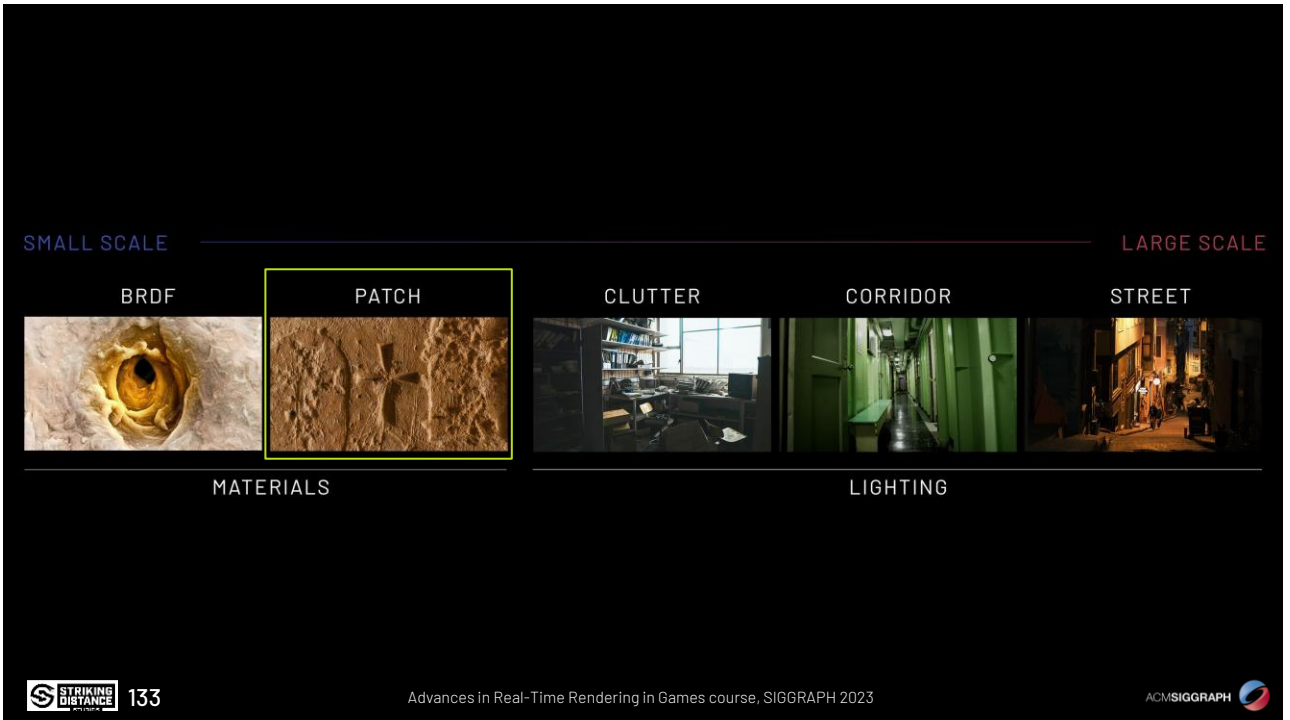
- α here has a different meaning to the α in other slices
- For numerical stability we recommend to pre-multiply by $\overline{\cos \theta_i}$ to remove the division

Here we finally have the math of how we combined Proxima and Callisto BRDF.



All this is great for rendering the balls that we often use to experiment with shading techniques.

But some objects are a bit more complex than that.



Going back to our photorealism scale figure, events that happen at patch level play a huge part in the realism of an image.

From texture authoring errors, to micro shadowing and global illumination.

Every surface point has a different look, and it would be perhaps naïve to think that we can represent all of them in the same way.

REALITY

To match reality, we often take a start in the statistical models that PBR give us.

REALITY

SIMULATION OR STATISTICAL PBR
RENDERING

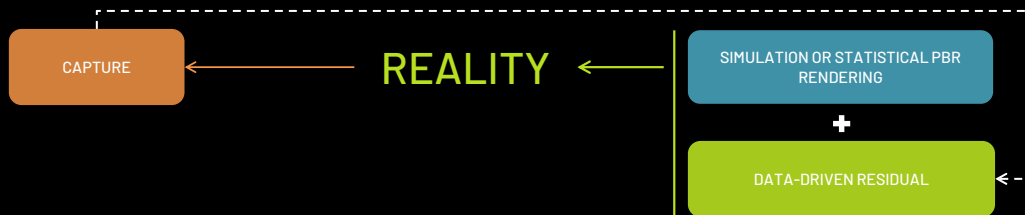


While they offer a very solid foundation, they make assumptions and have limitations.

Getting to reality with them is difficult, and I think it will take years or perhaps decades to fully understand the path to get there.

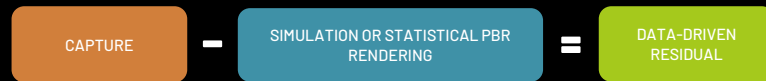


However, reality is readily available for us to capture, today.



Our proposal is then, why we do not try to get as close as possible with state-of-the-art PBR techniques, and then bridge the gap with a data driven residual.

REALIS



This residual is the error from what we capture using highly controlled photography, and standard rendering techniques.

This is where Realis, our technology to reproduce reality takes the spotlight.

It calculates differences from photographs to renders and bakes them into a database.

This data is then used in the runtime to reconstruct the shading error for a given light direction.

[Mostly for diffuse corrections]

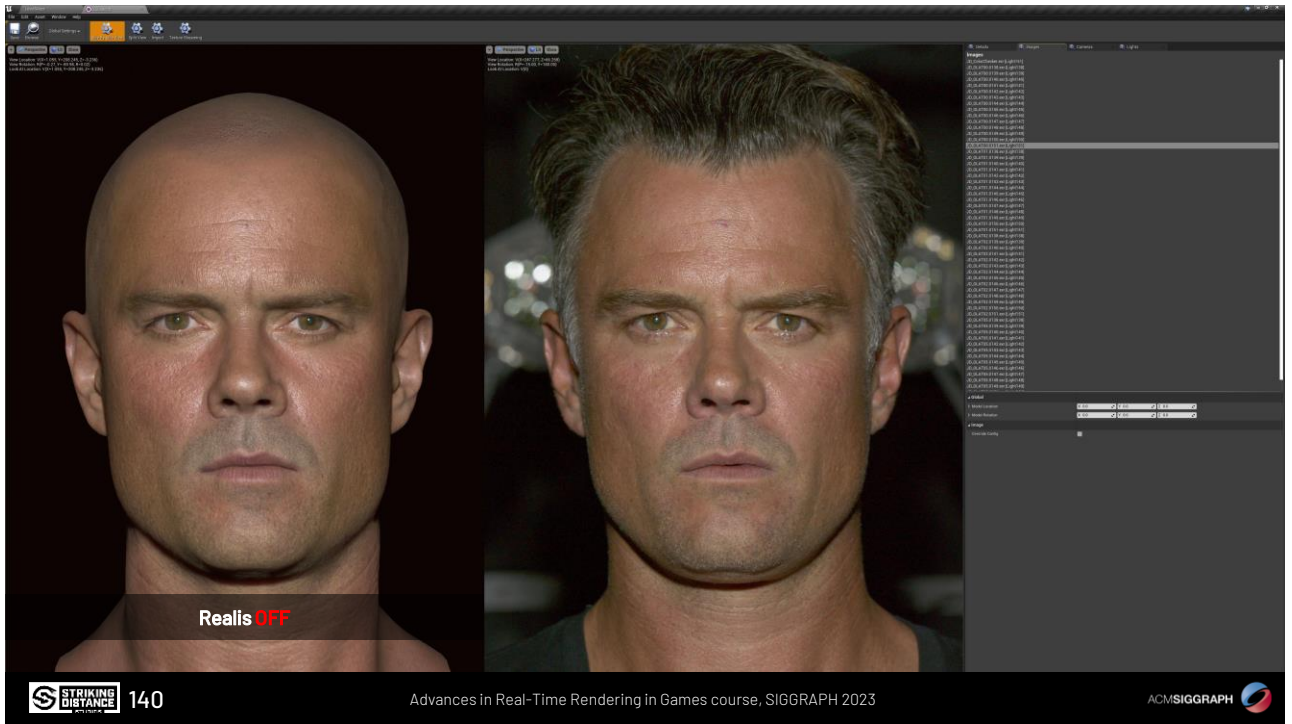
POLYNOMIAL TEXTURE MAPS

- Capture a patch under multiple lighting conditions
- Project complete captured lighting into a basis
- In our case we project the error



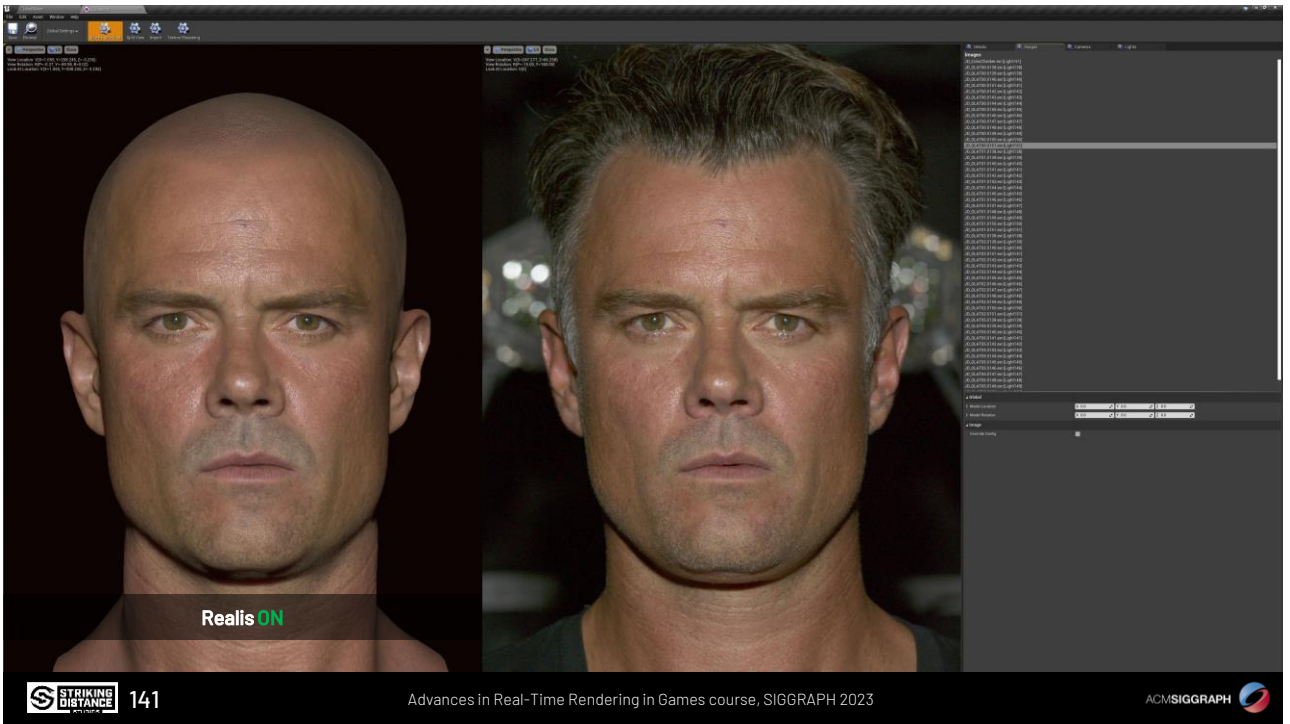
This idea is not dissimilar from polynomial texture maps, but in that case they bake the entire lighting into a basis, creating severe interpolation errors as light direction changes.

In our case only the error interpolates, so those errors are harder to notice.

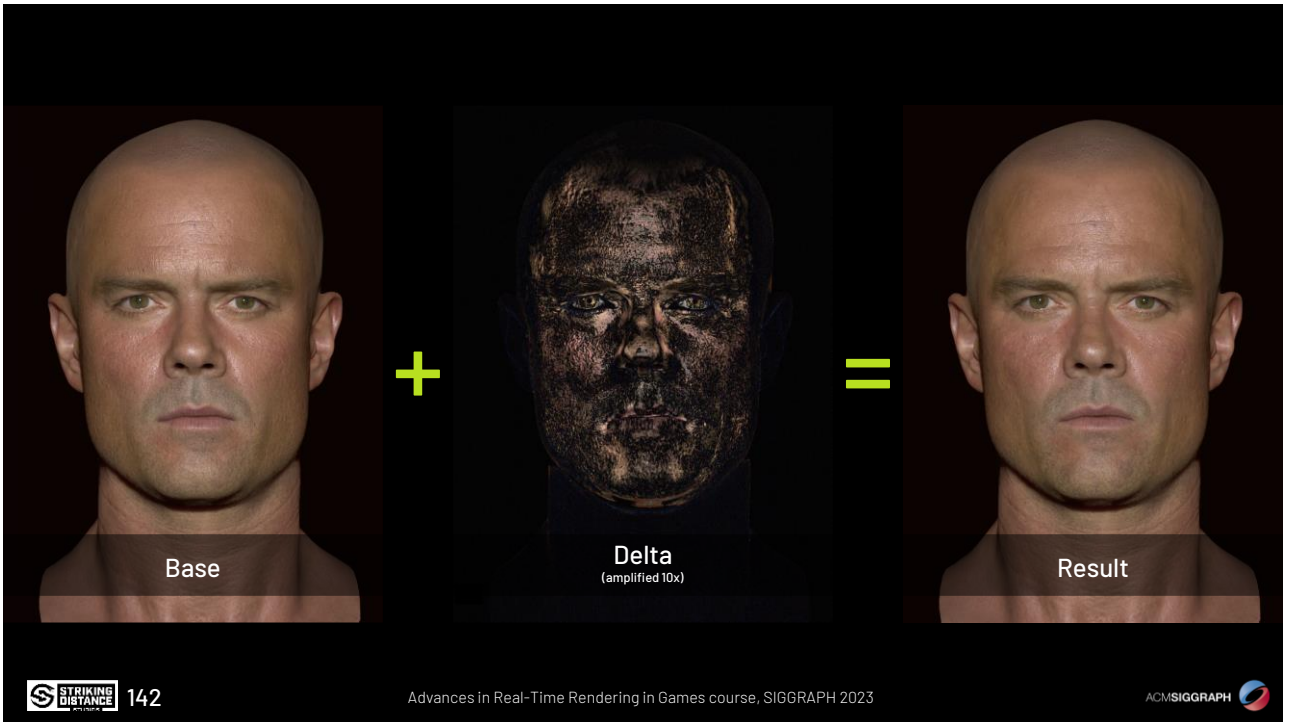


So here we have what realis does.

Everything that we were unable to capture in art, Realis brings it back from the photograph.



[back and forth]



Conceptually, it takes what we can't do in realtime, and applies it back in the runtime.

Miguel will dive later into the details of how this was implemented.

Ground truth and consistent lighting.

To finalize our journey towards photorealism, lighting consistency gets as much weight as materials.

SMALL SCALE

LARGE SCALE

BRDF

PATCH



MATERIALS

CLUTTER

CORRIDOR

STREET



LIGHTING



144

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023



To achieve photorealistic results during gameplay all lights have to behave in the same way.

All the attention put into a material will be lost, if it does not behave the same way under different type of lights.

LIGHTING CONSISTENCY GOALS

- Defining **consistency**:
- Using **full class lights**:
 - Stationary (most accurate)



Unreal offers static, stationary and movable lights, which create a different look on the materials.

Stationary lights are the most accurate ones, so we wanted them to be the foundation of the lighting in the game.

Unreal Engine only supported 4 overlapping stationary lights, which was not enough for our game, so we first extended that to 16.

LIGHTING CONSISTENCY GOALS

- Defining **consistency**:
- Using **full class lights**:
 - Stationary (most accurate)
 - With shadows enabled
 - Without aggressive attenuation
 - Fully featured over raytracing, volumetrics, transparents (IES Profiles, Material Functions, etc.)



More importantly, we made all lights full class.

That means they have shadows, avoid aggressive attenuation where possible, and are fully featured, in that they will look identical in raytracing, volumetrics and transparent surfaces.

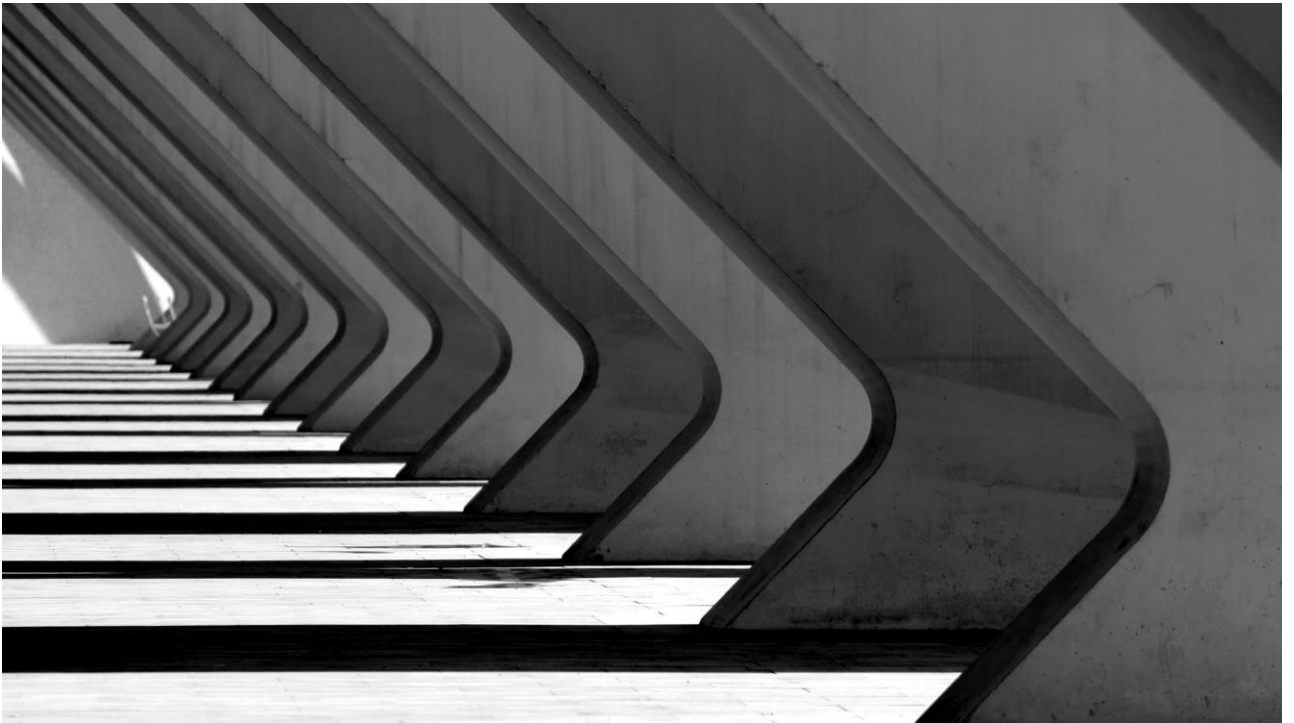
LIGHTING CONSISTENCY GOALS

- Defining **consistency**:
 - Using **full class lights**:
 - Stationary (most accurate)
 - With shadows enabled
 - Without aggressive attenuation
 - Fully featured over raytracing, volumetrics, transparents (IES Profiles, Material Functions, etc.)
 - Using **robust foundation**:
 - Raytraced shadows
 - Raytraced reflections

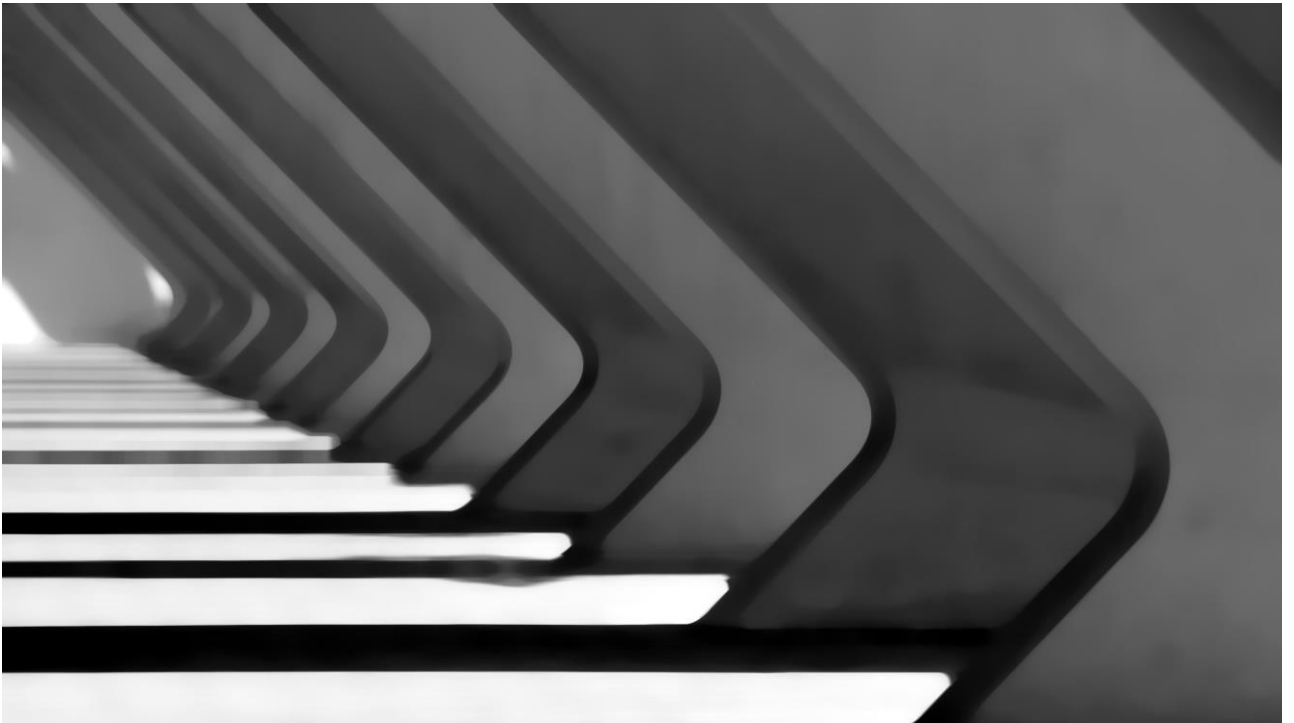


We also wanted lighting to have a robust foundation.

For us, this meant investing in raytracing.



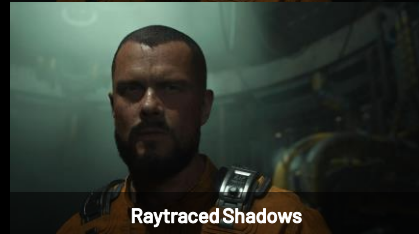
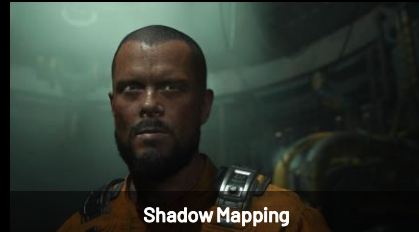
From my point of view, shadows are the most important salient feature of many images, which is specially true for horror videogames.



They contain the broad strokes and foundation that will give support for the rest of the details to follow up.

RAYTRACED SHADOWS CONSISTENCY

- Light leaks are most dangerous enemies of photorealism
- Shadow mapping is not consistent
- Raytracing:
 - Is robust
 - Can be on ballpark of raster if using caching
 - Unlike raster, scales efficiently with screen size
 - Allowing for large amounts of shadowed lights if small



The usage of shadow mapping heavily limits the quality of the results, if we look at the problem from a consistency perspective.

Raytracing in the other hand is robust, and as Miguel will show later, can be performant if using caching and other techniques.

RAYTRACED REFLECTIONS CONSISTENCY

- Traditional techniques limited by screen info and baked data
 - Light leaks on probe reflections
 - Objects not always grounded



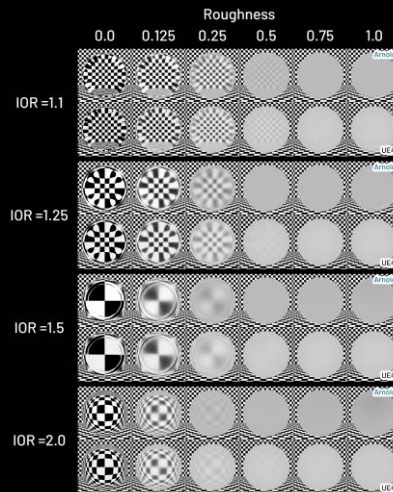
Similar thoughts apply to reflections.

Traditional screen space techniques came with significant artifacts in form of either over occlusion or leaking.

On the other hand, raytracing allows to ground all the scene elements correctly regardless of their screen space complexity.

GROUND-TRUTH GLOSSY REFRACTIONS

- Goal: Have proper light refraction behavior in specific surfaces (like curved glass)
- For performance reasons, we used **Screen Space** for the traces
- **PBR based** implementation
 - Renders in Arnold as ground truth
 - Rays follow **GGX** distribution function
- Pure ground truth solution in screen space had inherent issues
 - We had to add workarounds



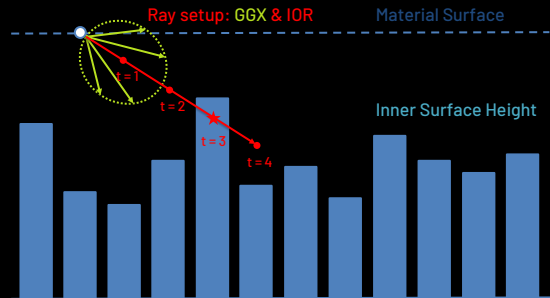
We also implemented a screen-space glossy refraction technique, trying to stay as close to the PBR ground truth as possible.

For that we compared with Arnold to evaluate the results.

But unfortunately, we had to take some shortcuts to address some of the inherent issues coming from tracing in screen space.

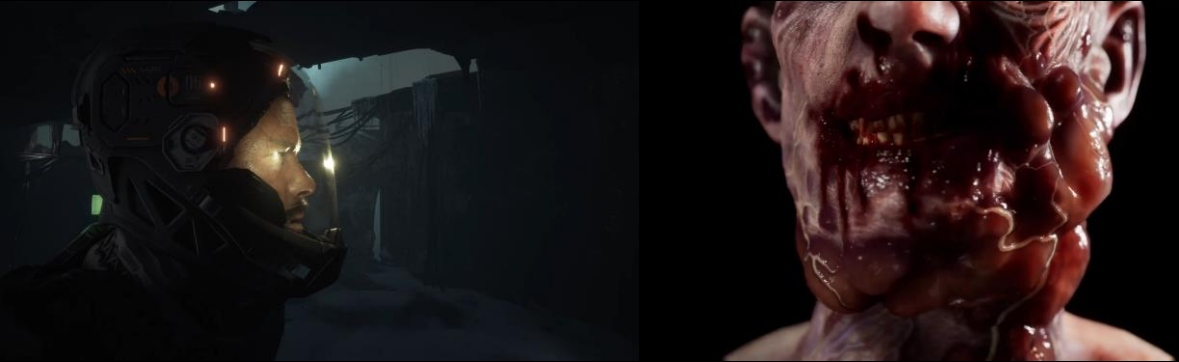
GROUND-TRUTH GLOSSY REFRACTIONS

- For refraction within **Materials** (Tissue):
 - We opted for a **Texture Space** approach
 - Revealing inner layer within the surface
- Same ray-marching logic as the Screen Space Refraction system
- Using heightmap of the inner layer height/depth
- Similar to Parallax Occlusion Mapping
- Supported flipbook & reveal mask animation for a variety of effects



To also have refraction in tissues we implemented a texture space version of the technique, which works in a similar way to parallax occlusion mapping.

GROUND-TRUTH GLOSSY REFRACTIONS



(Developed by Hampus Siversson, Edu Sanchez and Miguel Rodriguez)
Advances in Real-Time Rendering in Games course, SIGGRAPH 2023



These techniques were used to render the visors and translucent tissues in creatures.

LIGHTING CONSISTENCY GOALS

- Defining **consistency**:
 - Using **full class lights**:
 - Stationary (most accurate)
 - With shadows enabled
 - Without aggressive attenuation
 - Fully featured over raytracing, volumetrics, transparents (IES Profiles, Material Functions, etc.)
 - Using **robust foundation**:
 - Raytraced shadows
 - Raytraced reflections
- Supporting **large** numbers of lights

155

Shortcuts were not an option given our vision, but consistency comes at a high price.

So we had to find our way to make a large number of shadowed lights with long attenuation radiuses fully performant.

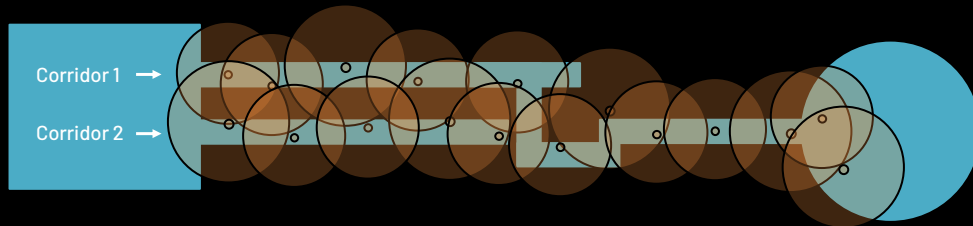
THE PROBLEM: CORRIDORS



Render



Photo



If we recall our game genre, it was linear in nature, with long corridors, side paths and rooms.

This means light overlap can be very high and we might be rendering more lights than needed.

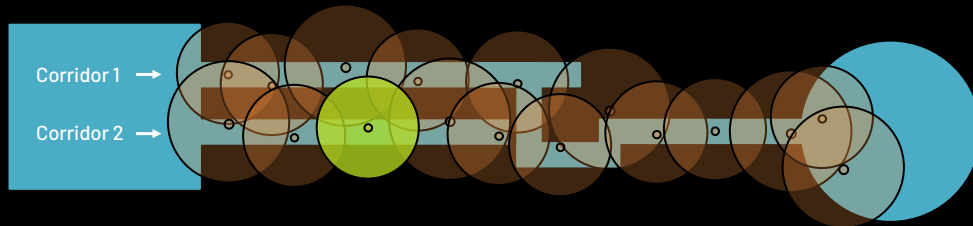
THE PROBLEM: CORRIDORS



Render



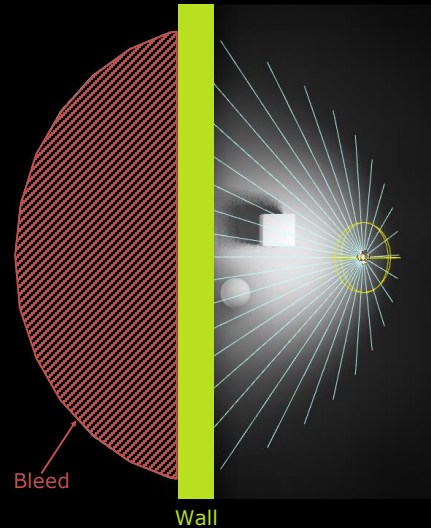
Photo



In this case for example the light marked in green will still be rendered from the corridor 1, even if it only has effect in corridor 2.

HULL-BASED LIGHT CULLING

- UE4 uses occlusion queries
- Lights evaluate/bleed across walls
 - Even when not visible
 - Performance loss
 - Artificially increases overlap on stationary lights



(Developed by Andy Yelland, Jay Ryness and Jose Naranjo)

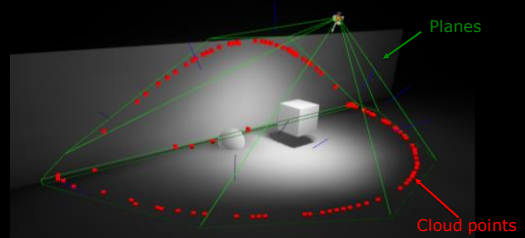
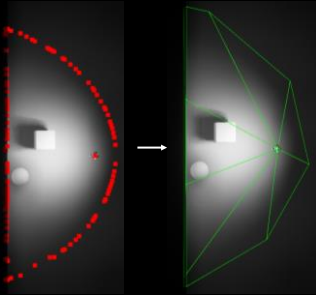
Advances in Real-Time Rendering in Games course, SIGGRAPH 2023

Unreal Engine uses occlusion queries, which use the full shape of the light, but not accounting for shadowing.

In this case it would render from the left side of the wall, even if the light would not be visible from that location.

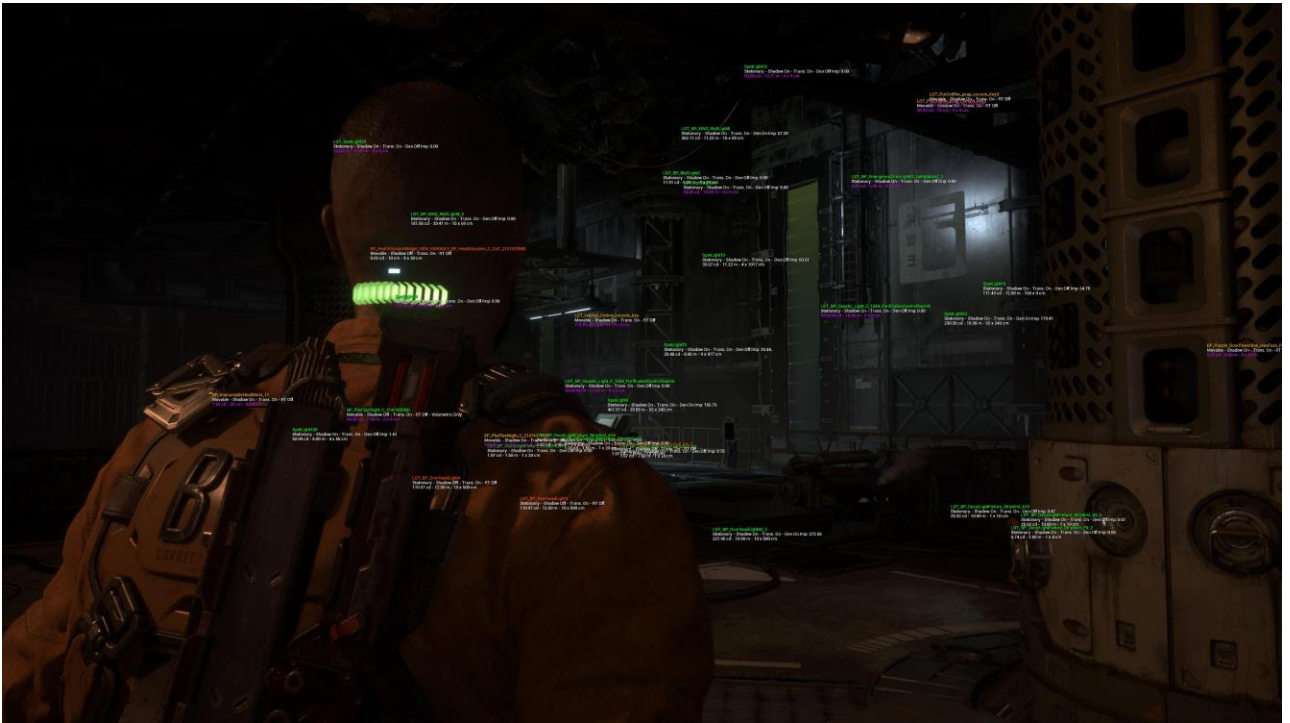
HULL-BASED LIGHT CULLING

- Fitted Light Hulls during light bake
- **Point clouds** around the shadow edge
- Points converted into fitted **sets of planes**
- Light Hulls used for:
 - CPU light rejection
 - Classic deferred lighting bounding
 - Forward grid tile culling
- **Allowed us to reduce overlap**
 - **~2ms perf improvements**



To address this, we fit light hulls during light bakes using a set of bounding planes.

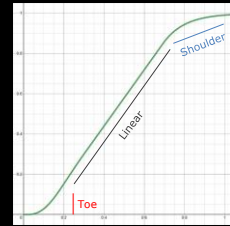
This allows for the lights to be constrained to the corridors and rooms where they belong, drastically reducing the number of lights required to be rendered.



In this scene we were able to render up to 45 shadowed lights, most of them using raytracing.

EXPOSURE ZONE SYSTEM

- Gran Turismo “GT Tonemapper”
 - Has a linear part
 - Helps to portray photorealism
- Zone Exposure System
- Used to calibrate the final frame



We used the Gran Turismo tonemapper to further our pursuit of photorealism.

The motivation for this decision is that it has a linear portion that we believe helps to convey natural and believable results.

We used the Zone System, a photographic technique used to determine optimal film exposure, to help guide the process of selecting the proper dynamic range for the game.

We added a mode to overlay multiple grayscale tones on top of the screen, where each one is one stop more intense than the one before.

The original Zone System defined 11 zones, but we used 17 instead more to account for high dynamic range (HDR).

We also added the ability to color code the image so that the toe, the linear and shoulder parts of the tonemapper are clearly displayed.

Observing the world.

Observing the world is crucial in the pursue of photorealism.

Given the time constraints I will skim over this section but still give an overview.

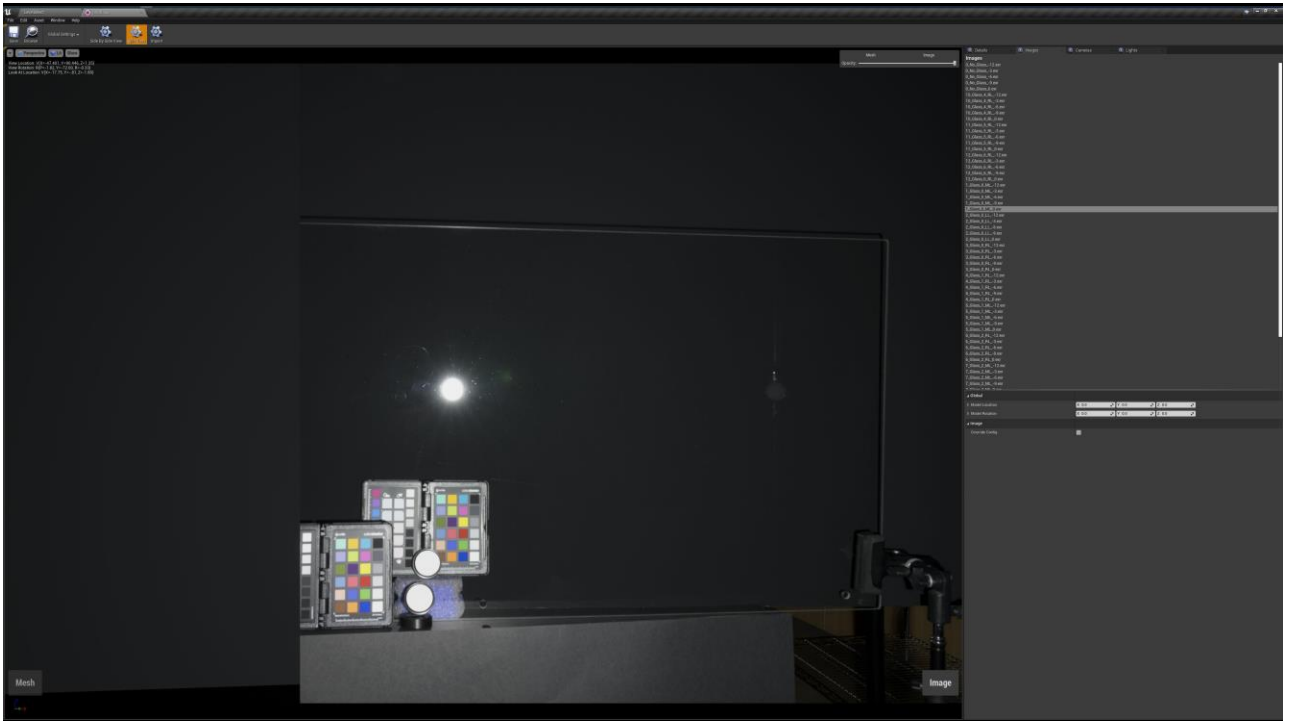
LIGHT RIG



This took multiple forms in our project.

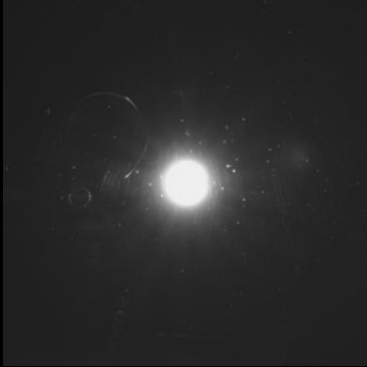
One of them was building a simple light rig for capturing photographic reference.

For example the prisoner suit...



...and glass.

BLOOM DIGITAL DOUBLE



Reference



Reference With
Radial Blur



Render



We also used this to fit the bloom in our game to match photographic reference.

RIVER DIGITAL DOUBLE

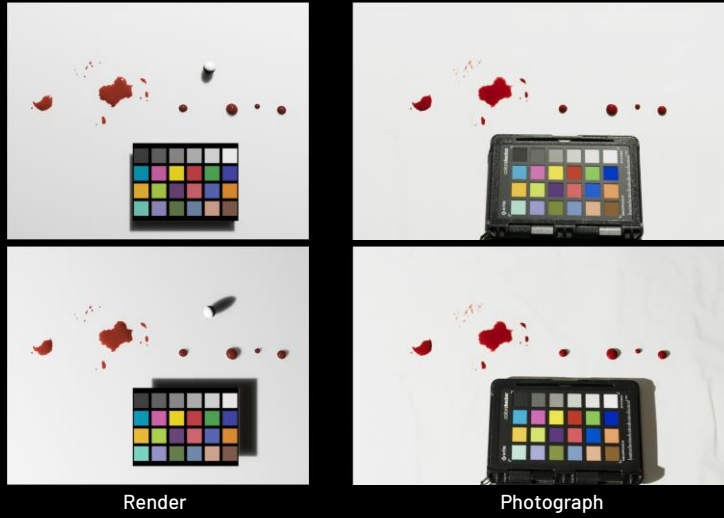


Here we have another digital double that we built to learn about water visual mechanics.

We got out of the office to film the Ebro river in Zaragoza, to then create a loose digital double of it.

What we learnt was applied directly into the execution of the pipeslide in our game.

BLOOD DIGITAL DOUBLE



(Developed by Pablo J de Andres)

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023

Here we have a digital double of blood drops, compared with photographic reference.

This experiment allowed to learn insights about blood appearance, as we are about to show.

OUR BLOOD COLOR STUDY

| BLOOD DROP | TIME | | | | | | | | |
|------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| | 00:00:00 | 00:02:00 | 00:05:00 | 00:10:00 | 00:15:00 | 00:30:00 | 00:45:00 | 01:00:00 | 03:00:00 |
| 01 | | | | | | | | | |
| 02 | | | | | | | | | |
| 03 | | | | | | | | | |
| 04 | | | | | | | | | |

(Developed by Pablo J de Andres)

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023

Gore and blood is an important element for our game.

What is color of blood is not an easy question to answer, as there are many valid colors.

The color depends in the amount of oxygen in the blood.

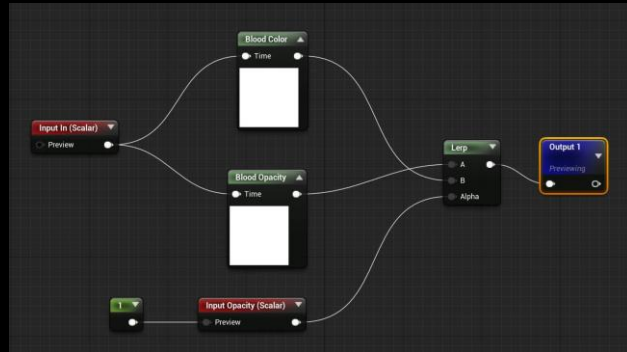
That means depending on where the blood comes from, and how much time has been exposed outside of the body, the color will be brighter or darker.

To find a valid range of colors, we captured blood drops over the course of 3 hours, for different thicknesses, and recorded the results.

OUR BLOOD COLOR STUDY

| COLOUR FROM DATA TABLE | COLOUR FROM MATERIAL FUNCTION |
|------------------------|-------------------------------|
| | |
| | |
| | |
| | |

| OPACITY FROM DATA TABLE | OPACITY FROM MATERIAL FUNCTION |
|-------------------------|--------------------------------|
| | |
| | |
| | |
| | |
| | |



Material Graph

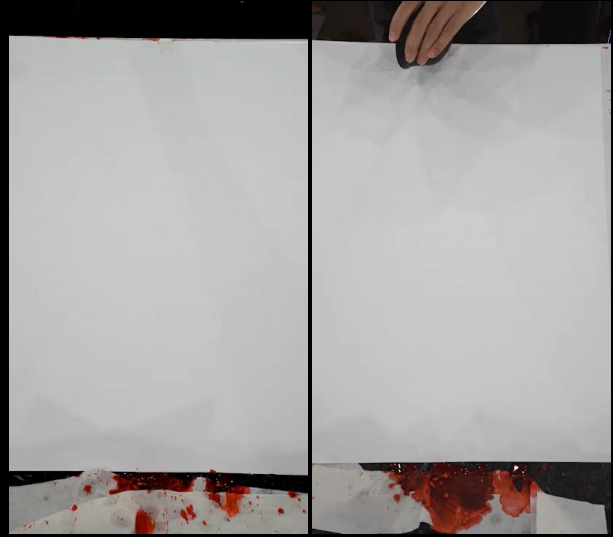
(Developed by Pablo J de Andres)

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023

With this information we created a palette of realistic blood colors, and created a material function that would give you the corresponding blood color based on the opacity and the time.

PHOTOREALISM THROUGH TRADITIONAL VFX

- Blood is very important for our game and traditionally it has been simulated or eyeballed.
- **Technical Traditional VFX:** We wanted to approach it with a rigid forensic observation base and then use traditional VFX captures to achieve a photorealistic result.



(Developed by Pablo J de Andres)

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023



170



Color is only one metric.

But mechanics and motion are as important.

Rather than relying in simulation we used a more traditional vfx approach, capturing blood drips using a mixture of artificial blood and viscous paste made with almonds.

BLOOD STUDIES

- Blood has been studied by forensics for a very long time. They study blood spatter patterns to determine where it originated.
- We have the opposite scenario, we know where a spatter starts and we want to know the end result.



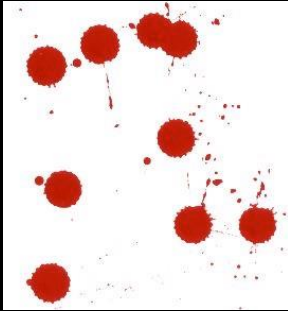
We also studied forensics to create realistic blood spatters.

They want to find where the blood came from, given a blood pattern.

We have the same problem but opposite.

We know where it comes from, but we want to determine the correct shape the resulting blood spatter should have.

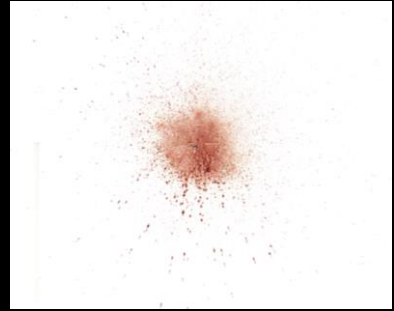
BLOOD STUDIES



Low Speed



Medium Speed



High Speed

The classification of forensics was a good start.

You have low, medium and high speed spatters.

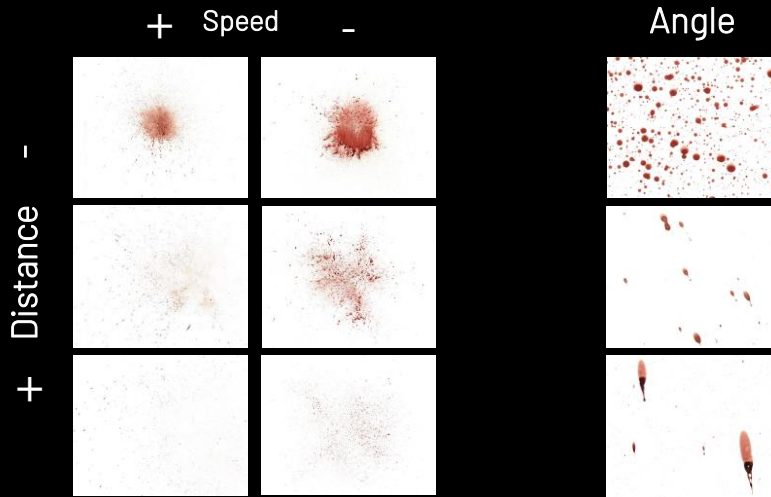
Low speed spatters are generated by gravity, and typically means a subject is moving after being wound.

Medium speed are caused by melee impacts.

And high speed by gun shots.

We followed a similar structure to give spatters the correct patterns according to speed.

BLOOD STUDIES



Daniel Attridge, Yu Liu, Ricky Falfiak, Yalin Rao, Bryce A. Struttman, Kris De Brabanter, Patrick M. Comiskey, Alexander L. Yarin, A data set of bloodstain patterns for teaching and research in bloodstain pattern analysis- Gunshot backspatters, Data in Brief, Volume 22 (2019), pp 269-278.

Similarly, blood spatters have different characteristics depending on the distance from source to receiver and the angle at which they arrive at the receiving surface.

BLOOD STUDIES

Forensic Captures



Daniel Attlinger, Yu Liu, Ricky Faflik, Yalin Rao, Bryce A. Struttman, Kris De Brabanter, Patrick M. Comiskey, Alexander L. Yarin, A data set of bloodstain patterns for teaching and research in bloodstain pattern analysis: Gunshot backspatters, Data in Brief, Volume 22 (2018), pp 269-278.

Our Captures



This is described and documented on forensic bibliography.

To develop our understanding of blood spatter behaviors and mechanics, we did experiments with artificial blood to create realistic patterns.

BLOOD SPATTERS

High Speed / Wall

Close Range



Medium Range



Long Range

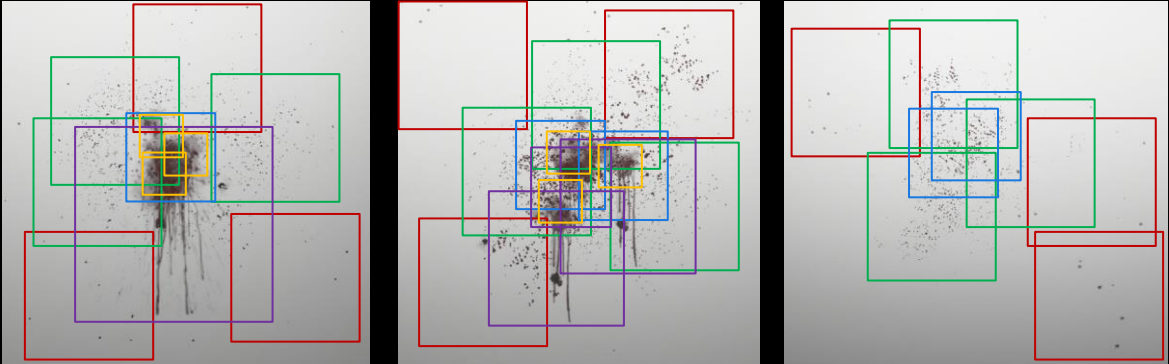


We used the captured information to create patterns that would react realistically to the distance from the blood source to the receiver surface.

BLOOD SPATTERS

High Speed / Wall

Core / Border / Outer Border / Mist / Drips



These patterns are composed of multiple type of decals: core, border, outer border, mist and drips.

When put together procedurally in the runtime, they create a look that represents well the observed appearance in the real world.

In that sense, each blood spatter in our game is different.

BLOOD STUDIES

Medium Speed / Floor

Close Range



Medium Range



Long Range



We followed a similar strategy for medium speed spatters...

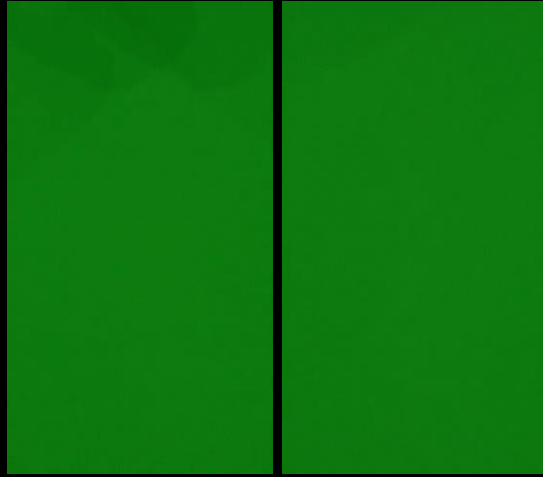
BLOOD STUDIES

Low Speed / Floor



...and low speed ones.

BRAIN DRIPS





Miguel Petersen

Senior Rendering Engineer, leading RayTracing.

[@miguel_oenp](#)



So that was the vision and theory, and I will now pass the torch to Miguel, for him to go over to the implementation, the technical challenges we found in our journey and how we managed to overcome them.

DEVELOPMENT METHODOLOGY

- We see Unreal as a stock implementation
 - Take what we need
 - Improve what we can
 - Replace what we must
- Insurmountable list of improvements not covered in this talk
 - Optimizations, visual improvements, etc.
 - But the good bits are here

We're here to make a game, not reinvent the wheel.

Unreal offers a fantastic base of support from which we can focus only on the bits we want to improve.

The topics covered in this talk is far from all changes, there are years worth of optimization changes behind the scenes, but the most impactful changes are here.

REALIS

Now, onto the first major feature, REALIS. And it's all about character rendering.



A lot of effort, truly.

REALIS

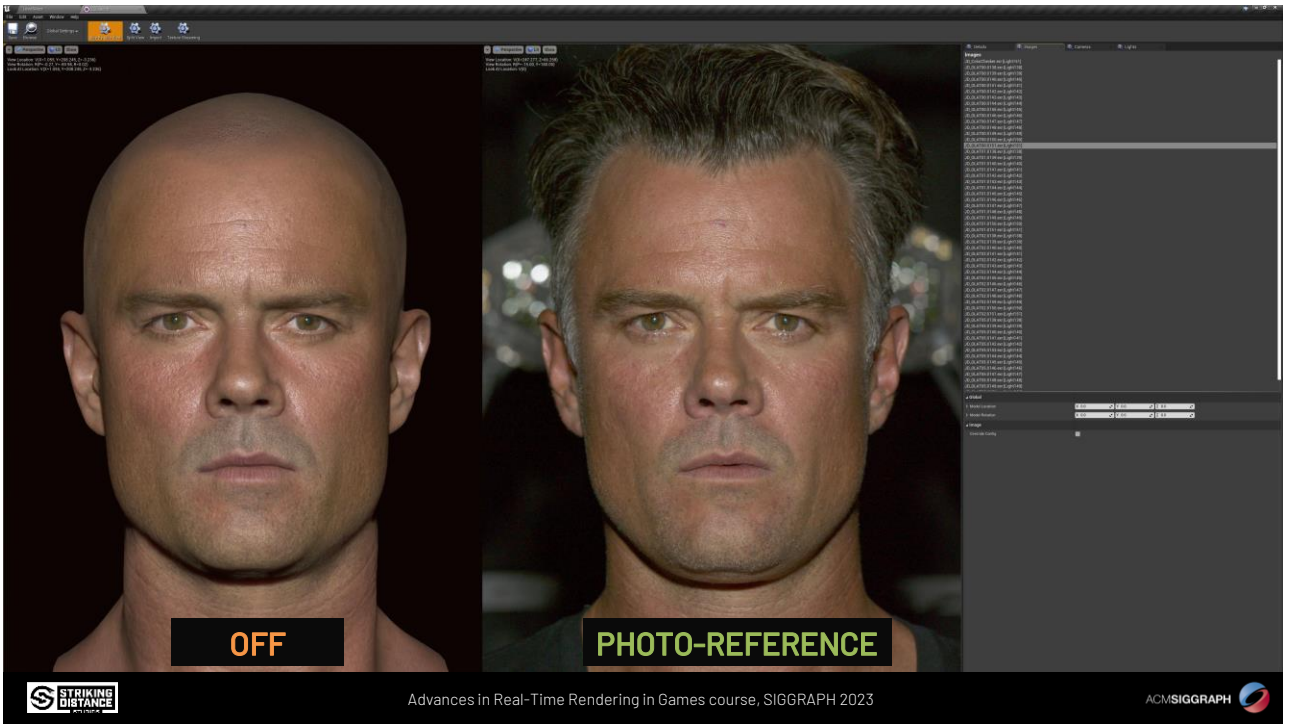
- State of the art rendering techniques are a marvel
 - Imperfect by nature
 - Realis is an acknowledgement of this
- BRDF error correction term
 - Multi-dimensional problem tied to BRDF

$$f_r(\omega_i, \omega_r)$$
$$f_r(\omega_i, \omega_r) + e(\omega_i, \omega_r)$$

So, state of the art techniques are a marvel, truly, but by nature they are imperfect.

We're approaching the problem bottom-up, and we're close. But, we wanted to try bridging that gap today.

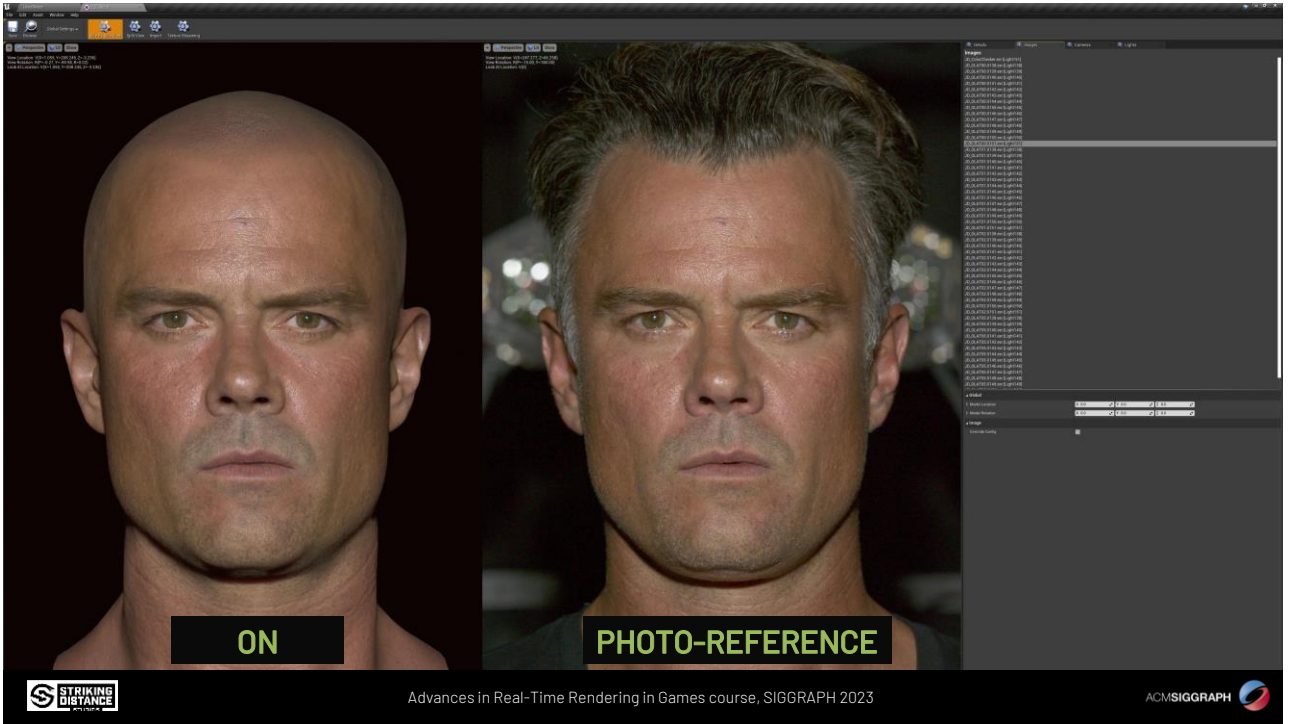
Enter Realis, at its core a BRDF error correction term. Whose initial dimensionality is tied to that of the BRDF.



And first, to sell the impact of Realis once more.

On the right you may find the photo reference, on the left is the in game render.

This is without Realis.



This is with realis.

REALIS

- Reduce dimensionality as much as possible
 - Diffuse only
 - Ignore **view** vector
 - Demodulate and reduce lighting parameters
 - Inverse lighting components invariant to incident

$$f_r(\omega_i, \omega_r) + e(\omega_i, \omega_r)$$

$$f_r(\omega_i, \omega_r) + e(\omega_i)$$

Now, excessive dimensionality is a bit of a problem.

We need to reduce it as much as possible, to do that we limited the scope to:

- Diffuse only
- View invariant
- Invariant to (most) lighting parameters and models

And from this we end up with the incident angle of the light.

REALIS

- Error term dimensionality
- RGB?
 - Full color is exceptionally close to reference
 - Memory and bandwidth requirements costly
- Luminance?
 - Close match, error to full-color acceptable
 - Memory reduced to $1/3^{\text{rd}}$

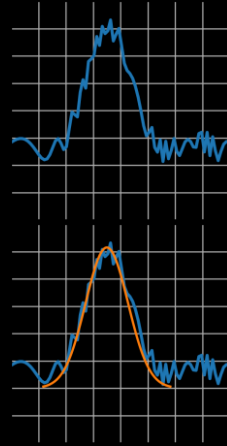
But, there's more.

What about the actual error itself, we saw two choices, either full RGB (three dimensions) or simply luminance.

From our tests luminance by itself was a very close match to the "accurate" RGB delta. We decided to go with this.

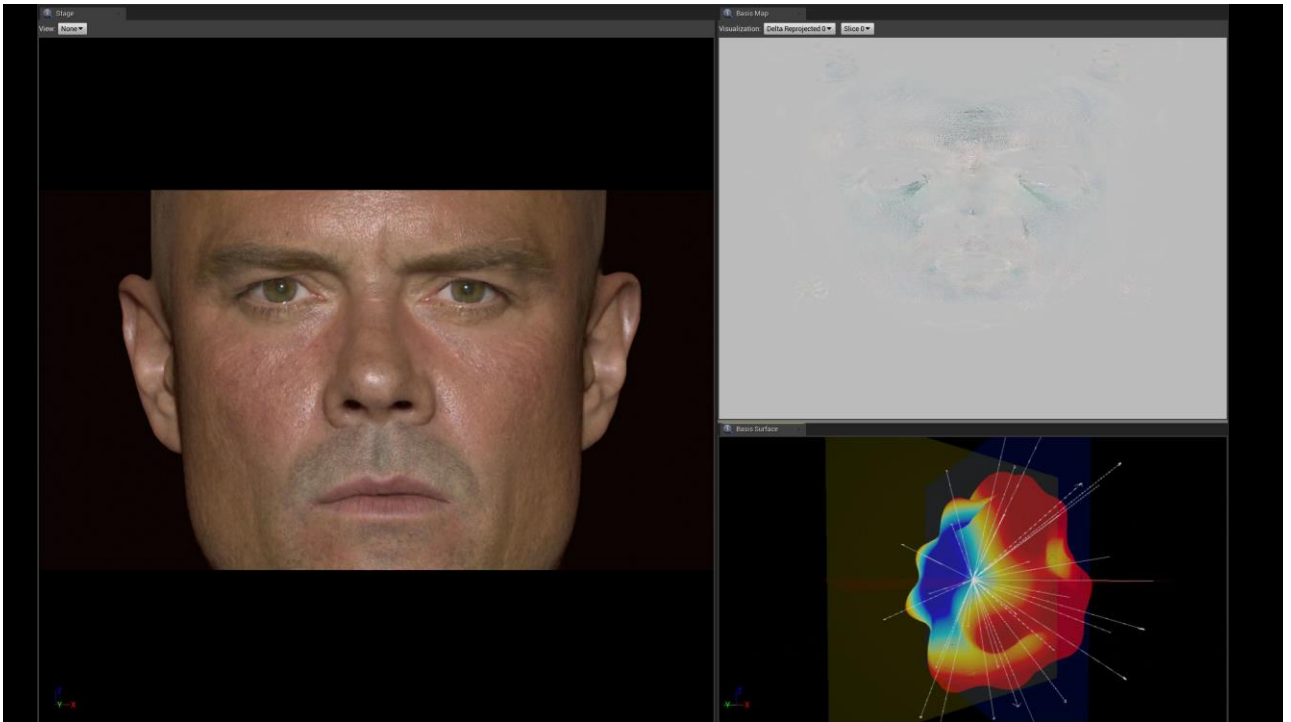
REALIS

- Represent light vector with spatial distribution
 - Windowed spherical gaussians
 - Model space
 - Non-uniform distribution
 - Sparse input data
 - Limited number of gaussians due to cost
 - Converge optimal distribution offline



We represent the error associated with a lighting vector through a spatial distribution, specifically spherical gaussians tied in model space.

This did not end up being a uniform distribution due to sparse input data, so the optimal distribution is automatically computed offline.



Here you may find our protagonist in an internal authoring tool, and on the bottom right you can see the final distribution chosen.

Red implies optimal coverage, we only wanted Realis contribution on the face, hence the hemispherical distribution.

REALIS

CAPTURE / BAKING



OFFLINE

G-BUFFER



RUNTIME

LIGHTING



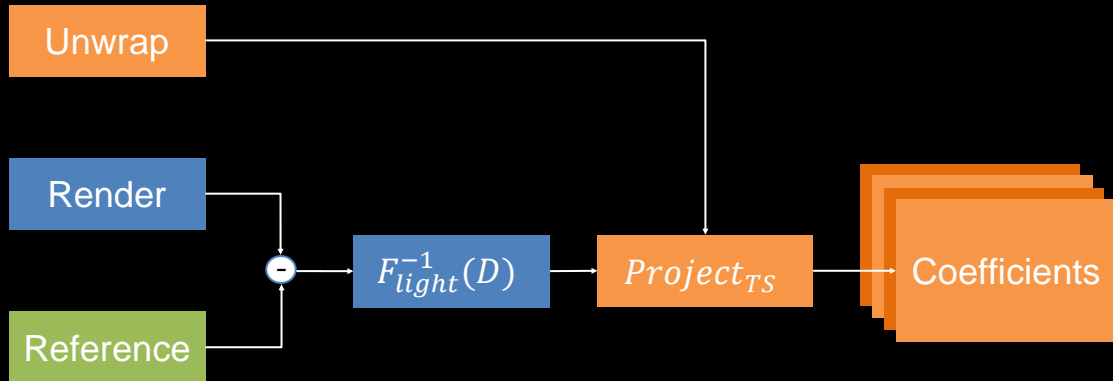
That's the idea, so, onto implementation. First, baking.

REALIS

Theory is simple, $E(D) = \textit{Reference}_D - \textit{Render}_D$

Well, should be simple, error is reference minus render.

REALIS



Let's visualize it.

- subtract the two
- invert the lighting function
- project it to texture space

and we'll have our coefficients!

REALIS

- Render data
 - Captured in *linear-space*
 - Before tone mapping
- Reference data
 - Color calibrated
 - Moved to *linear-space*
- That's it?

Rendering data is captured in linear space just before tone mapping, reference data is calibrated and then moved to linear space.

So, are we done?

REALIS

- Data is **imperfect**, and will always be
 - Camera noise
 - Foreign objects, render or reference
 - Geometric mismatch
 - Shadowing mismatch
 - Specular removal
 - Texture space stretching

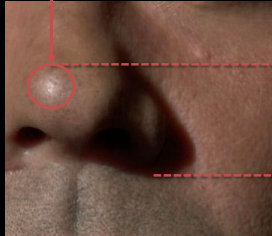
No, because data is imperfect, and will always be.

You have noise, foreign objects, geometric mismatches, shadowing is off, speculars are different, texture space projection stretching, error data is captured at a specific view, projection back to texture space can result in artifacts due to stretching.

Fun stuff.

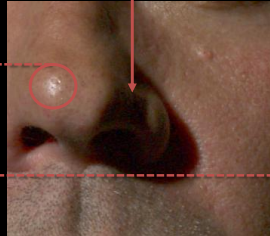
REALIS

Specular Mismatch



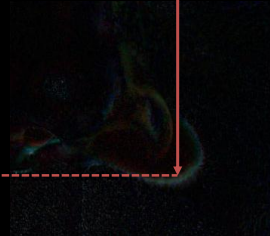
RENDER

Camera Noise



REFERENCE

Shadow Mismatch



DELTA

For example.

Plenty of camera noise here.

Shadows are not entirely aligned, naive baking would result in false errors.

Speculars aren't exactly the same.

This was a challenge to address, for the purposes of this session we are not going over the exact implementation specifics, but you may find some hidden slides on this.

REALIS

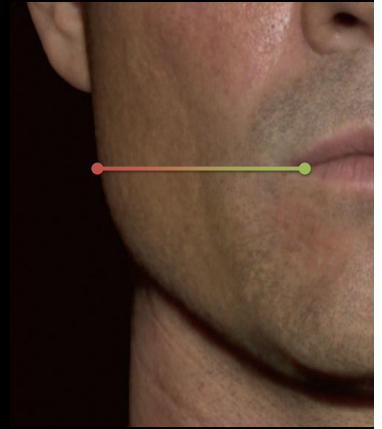
- Weight out large projection gradients
 - UV space stretching prone to artifacts
 - Compare view vectors against gradient

$$f = \left\| \frac{dP_{TS}}{dP_{CLIP}} \right\|^a$$

$$w = f * b * (1 - \|V_{camera} \cdot V_{lobe}\|)$$

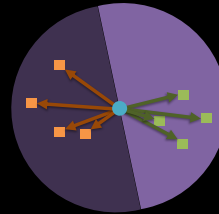
a : Gradient strength

b : Fade speed



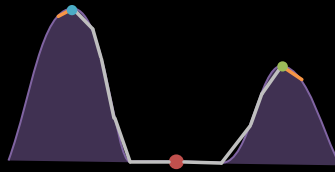
REALIS

- Shadowing mismatch
 - Detect shadow borders
 - Well, one part's dark, one part's not
- Average texel center to neighboring
 - $$V = \frac{1}{n} \sum \|T_{neighbor} - T_{center}\|$$
 - Two averages, one for shadowed areas and one for lit areas
 - Weighted threshold
 - Dot product represents the likelihood of a shadow contour



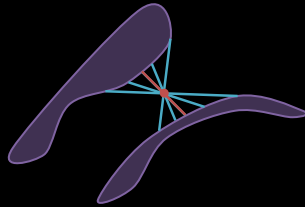
REALIS

- Two shadow contours
 - Reference contour
 - Render contour
- Mismatch shadow area is the gap between the two



REALIS

- For a given texel, do a spherical line trace
- Inbetween a contour?
 - Reduce coefficient weight
 - The larger the contour distance the bigger the problem



REALIS

CAPTURE / BAKING



OFFLINE

G-BUFFER



RUNTIME

LIGHTING



So, let's move on to the runtime.

REALIS

- Model coefficients exported during gbuffer rendering
 - No bindless support at the time
- Memory and bandwidth
 - Highly compressed coefficients
 - Offline precision multiplier to make best use of the bits
 - Temporal jittering
 - Reduce dataset by exporting relevant coefficients
 - Per-textel dominant light source mask



At the time we did not have bindless support, so we decided to export the relevant data during gbuffer rendering.

This is potentially a lot of data, in the interest of memory and bandwidth this was heavily compressed.

REALIS

- Source data is imperfect, runtime data is a lie
- Artistic changes on content due to gameplay
 - Dirt
 - Grime
 - Blood

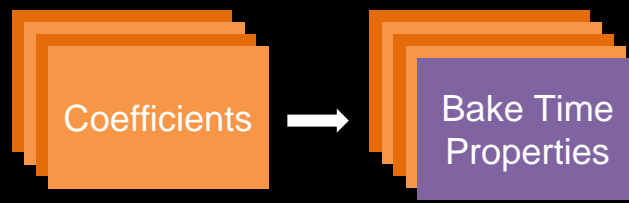


Unfortunately not, not only is the source data imperfect, but the runtime data lies.

There are plenty of gameplay events that can trigger changes to the material properties, such as albedo, normals, etc...

REALIS

- Realis error term captured at specific material properties / conditions
 - Artistic direction can invalidate this
 - Compute difference at runtime, weigh out contribution in affected areas



Problem is, the error term is exactly tied to the material properties at capture time.

To combat this we export a very thin version of the bake time properties, and then compare the runtime properties against the offline properties. If they differ greatly, fade out the realis contribution.

Not very complicated, but very important.

REALIS

CAPTURE / BAKING



OFFLINE

G-BUFFER



RUNTIME

LIGHTING



So now that we have the data, let's use it.

REALIS

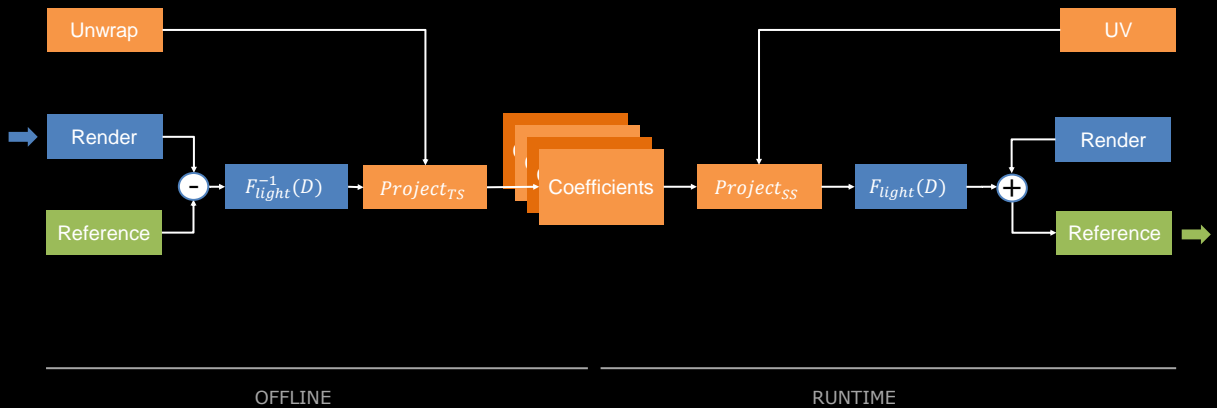
- Any given light may have Realis contribution
 - Project incident vector to model / skinning space
 - Evaluate gaussians from exported dominant mask
 - Re-apply lighting function
- Functional **inverse** of baking process

Every single light may have realis contribution.

To apply it, we first project the incident vector to model space. Then evaluate the relevant spherical gaussians, and apply the resulting contribution.

What is really pretty about this is that it's a functional inverse of the baking process. Let me show you.

REALIS



First, we subtract. Demodulate and invert the lighting function, and store our coefficients in texture space. That's all offline.

Then, at runtime, we technically project to screen space, modulate and apply the lighting function. Then add it to the current render.

We start with the render and end up with the reference. A functional inverse, exactly what we want.

REALIS

- End result is great
 - Primary drawback is geometric and texture alignment
 - Our character content pipeline assisted this drawback
- Further reduced memory and bandwidth with bindless support
- Problem space is a multi-dimensional feature fit
 - **Machine Learning** is the modern day solution
 - Being actively researched

And that's it. The concept is simple in nature, but its complexity comes from the data processing.

Our particular character content pipeline reduced the overhead associated, greatly, as it provided near perfect geometric and texture alignment.

And to the future, bindless for sure.

And, given the inherent nature of this problem, i.e. A multi dimensional feature fit, it really does scream machine learning. This is something we're looking at today.

RAYTRACED SHADOWS

We don't just want shadows, we want raytraced shadows.



Shadows on **everything.**

And we want it on everything, at every place, in every moment.

RAYTRACED SHADOWS

- Accurate shadowing is vital
- A difficult problem space
 - Typically numerous lights on the player and surroundings
 - Potentially hundreds of lights visible overall
- Pure raytracing, no hybrid rasterization

Shadowing is one of the most important effects to sell immersion, but it's a difficult problem space.

We typically have a bunch of lights around the player alone. Potentially hundreds overall.

And I cannot stress this enough, we follow a pure raytracing approach, there is no hybrid rasterization.



So, a challenge, but we're quite proud of the results.

RAYTRACED SHADOWS

- Unreal offers a great base
 - Material hit shader generation, shader record setup, etc...
 - The pipelining is there!
- Our needs greatly differed from the original intentions
 - Virtually all lights raytraced
 - Large screen coverage, up to 16 large lights at a time around you!
 - Because of this, we wrote our own raytracing implementation

Unreal offers a great base, all the relevant pipelining is there.

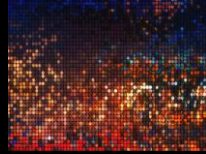
However, we found that our needs greatly differed from the original intentions, and types of content it applies for. So, because of this, we wrote our own raytracing implementation.

RAYTRACED SHADOWS

CACHING



VARIABLE



CULLING



SPECIALIZATION



DENOISING



TRANSMISSION



Real-time, raytraced shadows follows six pillars.

Caching, variable rate tracing, culling, specialization, denoising, and transmission.

RAYTRACED SHADOWS

CACHING



- Stationary lighting
- Offline cache
- Shadow projection masks
- Separate TLAS'es

Let's start with caching.

Going over stationary lighting, offline caching, shadow projection masks, and separate tlasses.

RAYTRACED SHADOWS

- Caching schemes easy with rasterization
 - Shared light-space
 - Depth compositing straightforward
- Difficult with raytracing
 - No shared sampling space



With rasterization caching schemes are easy, as a shared light space allows for simple re-composition.

However, with raytracing it's difficult as there is no implicitly shared sampling space.

RAYTRACED SHADOWS

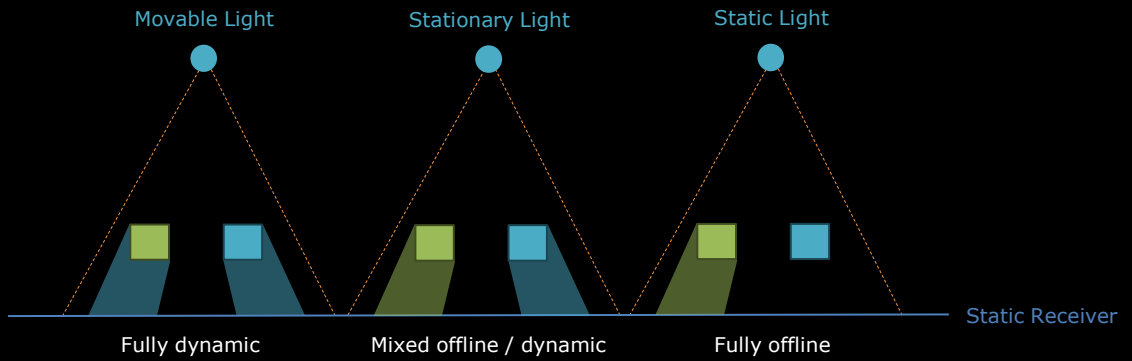
- First, a primer on Unreal lighting
 - **Offline**
 - Computation of static-on-static interactions
 - Volumetric computation of static-on-dynamic interactions
 - **Runtime**
 - Realtime computation of dynamic interactions
- Three light types
 - We avoid static lights

Before getting into it, just a quick primer on unreal lighting to set the stage.

Unreal offers both offline and runtime lighting, offline for static interactions, runtime for dynamic interactions.

And three lighting types within.

RAYTRACED SHADOWS



Static shadow / caster

Movable shadow / caster

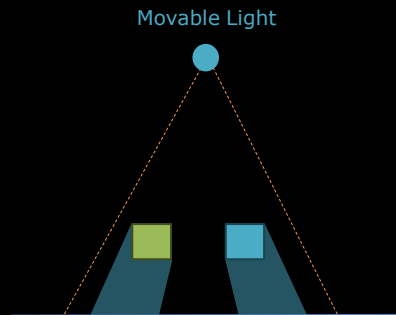
Movable, stationary, and static.

Please note that green denotes static interactions. And blue denotes dynamic interactions.

So, movable lights. [next]

RAYTRACED SHADOWS

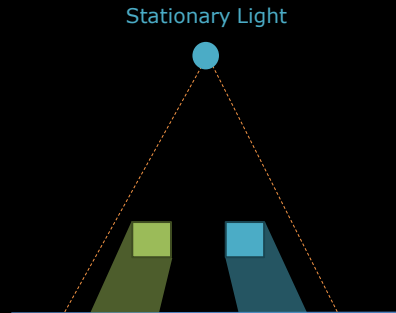
- Movable lights
 - Exclusively runtime shadowing
 - Runtime diffuse & specular



Movable lights offer exclusively runtime shadowing, and runtime lighting.

RAYTRACED SHADOWS

- Stationary lights
 - Offline shadows for static objects
 - Runtime shadows for dynamic objects
 - Runtime diffuse & specular



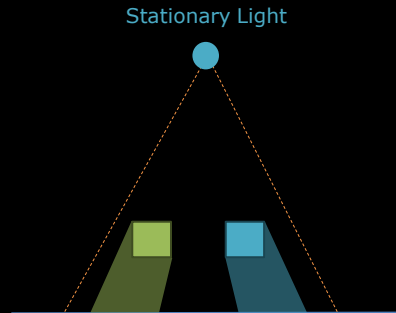
The second kind, stationary lighting, offers offline shadowing for static objects, and runtime shadowing for dynamic objects.

An interesting split, with a fully runtime direct lighting solution.

Please note that I am not going over static lighting, as that did not fit our use case.

RAYTRACED SHADOWS

- Stationary lighting fits
 - **Offline** static-on-static interactions
 - De-facto caching system
 - Lightmap space acts as shared sampling space
 - **Runtime** dynamic interactions
- Future, fully runtime caching
 - Use offline cache as a starting point
 - Refine over time based on sampling needs

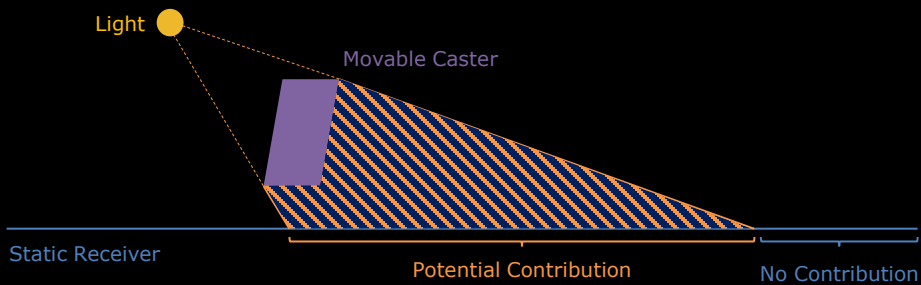


In comes stationary lighting.

The offline static interactions becomes the de-facto caching system in lightmap sampling space. It's a perfect fit.

RAYTRACED SHADOWS

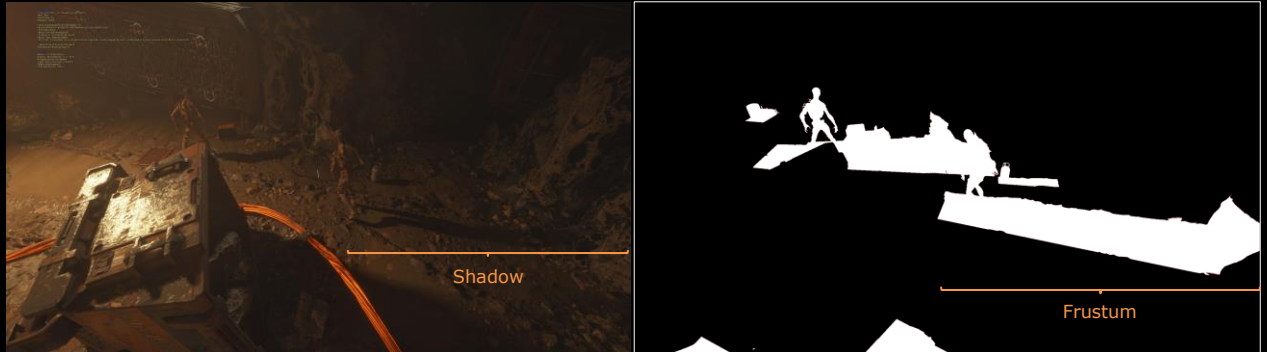
- Dynamic on static contribution can be reduced
 - Conservative estimation by projecting the caster AABB from light
 - Parallels to recent paper [Sammy Fatnassi 2023, Shadow Techniques from Final Fantasy XVI]



With stationary lighting, we can also make assumptions based on movable projections.

By projecting the effective caster frustum from the light, we can determine where we may find potential contribution, and most importantly where no contribution is possible!

RAYTRACED SHADOWS



To show this in practice, on the left you may find an enemy projecting a runtime shadow on the floor.

On the right you may see the effective caster frustum. We only perform runtime tracing inside that.

[Readers Note]

The implementation is a retro-fit of the stock-UE4 shadow projection masks for the rasterized counterpart.

RAYTRACED SHADOWS

- Separate acceleration structures
 - **Static & Dynamic TLAS**, static on dynamic shadowing
 - **Dynamic TLAS**, dynamic on static shadowing



The separation of offline and runtime shadowing allows for an interesting optimization.

We have two TLAS's, one for static and dynamic objects, and one for just dynamic objects.

Dynamic objects may receive both shadowing from both static and dynamic objects.

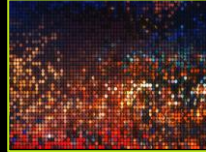
However, static objects already have the static interactions precomputed, so, we just need to trace the dynamic counterpart, a vastly simpler acceleration structure.

RAYTRACED SHADOWS

CACHING



VARIABLE



CULLING



SPECIALIZATION



DENOISING



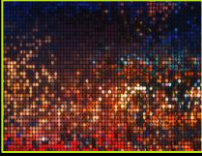
TRANSMISSION



So, that's caching, now onto variable rate tracing.

RAYTRACED SHADOWS

VARIABLE



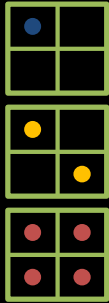
- Variable rate raytracing
- Shadow variance
- Perceptual weighting
- Bilateral upsampling

Which consists of, the actual variable rates, shadowing variance, perceptual weighing, and bilateral upsampling.

RAYTRACED SHADOWS

Variable Rates

- Reduce sample count while maintaining quality



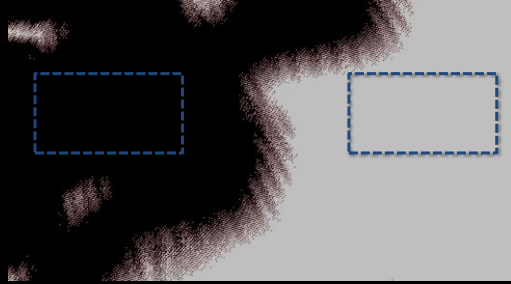
Quarter, 1:4

Half, 1:2

Full, 1:1

Intra-Quad Shadowing

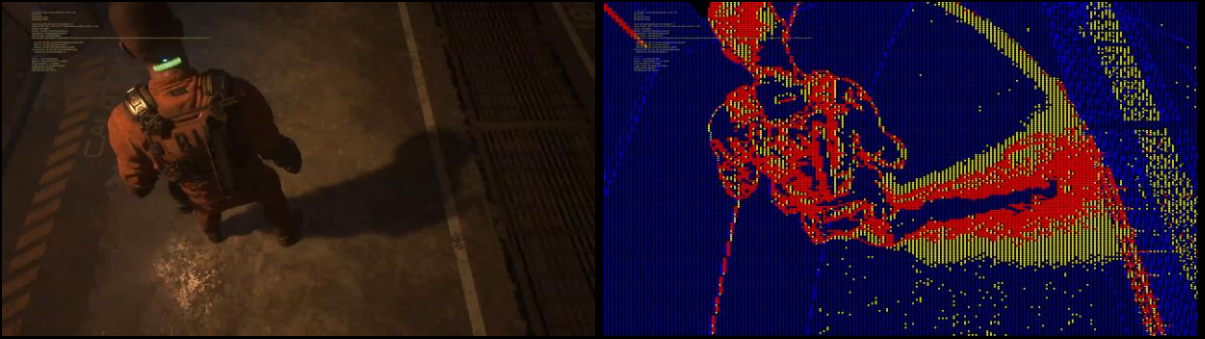
- Majority intra-quad shadowing can be reduced to **one sample**



The whole point is to reduce the sampling count while maintaining quality, to do this we enable three rates, quarter rate, half rate and full rate.

The majority of shadowing can be reduced to one sample, i.e. quarter rate, as seen in the example on the right.

RAYTRACED SHADOWS



As you can see in the video. What this means in practice is that we focus our samples on the silhouette of the shadows.

RAYTRACED SHADOWS

- No way to calculate the perfect rate before tracing
- Compute **variance** of shadow occlusion **after tracing**
 - Determine an appropriate rate given thresholds
 - Feed rates to **next frame**
 - Difficult to spot error, denoising will help



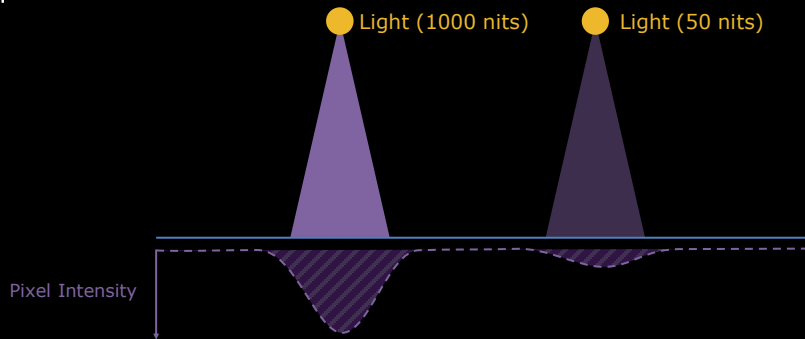
To determine appropriate rates we compute the variance of the shadowing occlusion after tracing.

This is then fed to the next frame.

We find it difficult to spot errors due to latency, especially as the denoiser will help any affected areas.

RAYTRACED SHADOWS

- Shadowing variance does not equate to **visually perceptive variance**
 - Low light intensity
 - Auto exposure
 - Bloom
 - Etc...



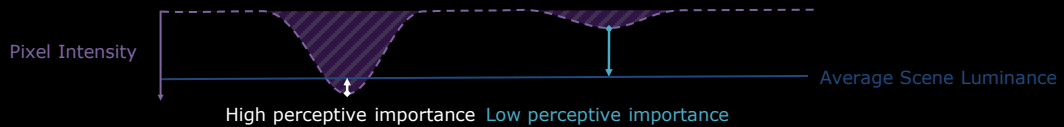
However, variance may not equate to the visually perceptive variance.

Lights may be less important due to low intensity, auto exposure, bloom, etc...

In the example we have one light with 1000 nits, and one with 50, and the pixel intensities associated. One clearly brighter than the other.

RAYTRACED SHADOWS

- **Weight** the rate based on the lights perceptual importance
 - During culling? Same problem, difficult to guess.
 - After **lighting**! Perfect information, - frame latency.
- Weigh rate based on tile luminance compared to average scene luminance
 - *Alternatively* could have worked in absolute tonemap space



To combat this we weight the rates based on the lights perceptual importance after lighting.

Implementation wise, we weigh the rates based on the tiles luminance compared to the average scene luminance. More on tiles later.

In the example below, with the same set of lights, the left light has a high perceptive importance, but the right light has a low perceptive importance when compared to the average scene luminance.

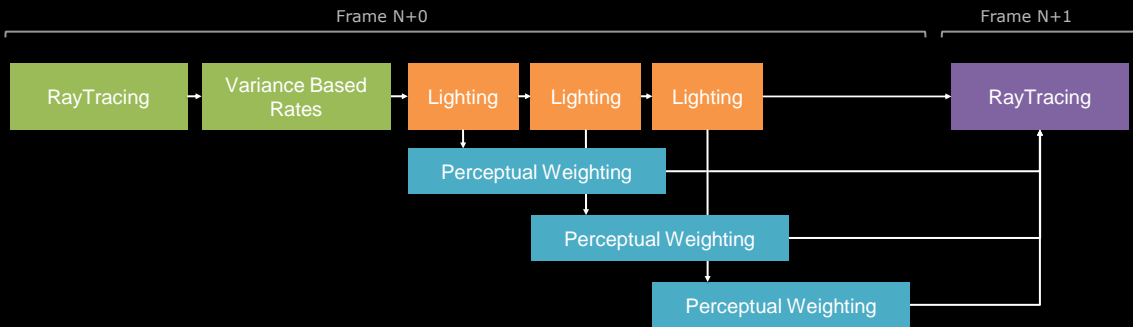
RAYTRACED SHADOWS



And to visualize it, first we trace, then compute the variance based rates, do the lighting, and only then weigh the rates on the perceptual importance.

All fed to the next frame.

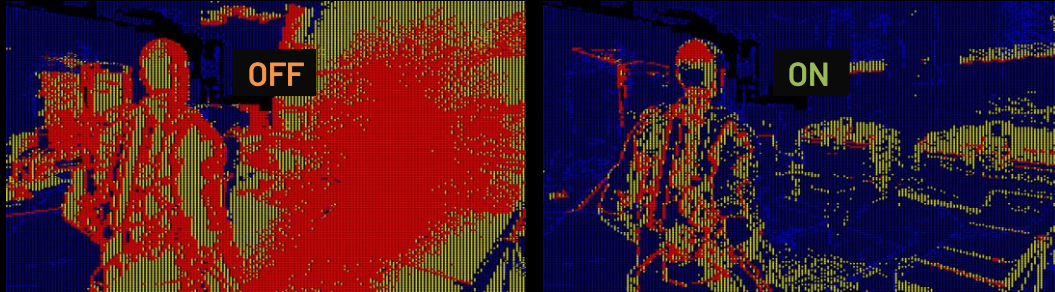
RAYTRACED SHADOWS



This is extended to any number of lights in the scene.

RAYTRACED SHADOWS

- Great improvements
 - May reduce more than half of the samples to one sample per quad
 - Heavily depends on the scene and its lighting conditions



In practice this works great. Left is without the feature, right is with the feature, a large reduction in samples.

This of course depends on the scene and its lighting conditions.

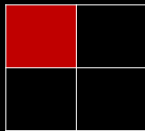
RAYTRACED SHADOWS

- Traditional scheduling
 - Work queues per variable rate with pixel wise offsets
 - Indirect parameters / signatures
- We did not ship a traditional scheduler
 - Early limitations made this difficult
 - Preferred path today
- Instead followed a “tile packing” mechanism

Traditional scheduling is apt, such as, a work queue with pixel wise offsets, followed by indirect parameters. However, we actually did not ship a traditional scheduler due to early limitations, and instead follow a tile packing mechanism. Due to time constraints, we will not be going over that today.

RAYTRACED SHADOWS

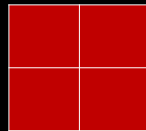
- Variable rates leaves **gaps**
- Patch holes with **bilateral upsampling**
 - Sampling positions before tracing
 - Reconstruction after tracing



¼ Rate



½ Rate



Full Rate



½ Res. MinMaxOff ½ Res. MinMaxOn

And of course, variable rates leaves holes in shadowing. So we need to patch them with bilateral upsampling.

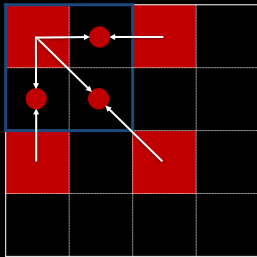
This is done in those phases, first, traces, then reconstruction.

Caution: the nature of the tile culler makes it so your lanes end up reshaped in your wave, be careful when reading your neighbours, they might not be sampling what you think.

RAYTRACED SHADOWS

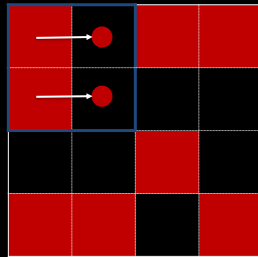
Quarter Rate

- Trace top-left
- Reconstruct by averaging with neighbors



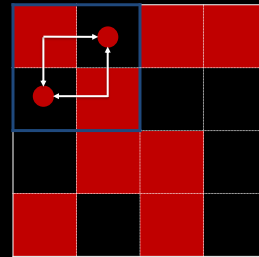
Half Rate: Borders

- Trace in the two most significant samples (MinMax Z)
- Reconstruct by copying into nearest neighbors



Half Rate: Flat

- Trace diagonally
- Reconstruct by averaging both samples



For quarter rate, we simply reconstruct gaps by averaging them with the neighbours.

For half rate on borders, we trace from the two most significant pixels in a quad. And then copy them into the gaps.

For half rate on flat areas, we employ diagonal sampling, and then average the gaps to fill the gaps.

RAYTRACED SHADOWS

CACHING



VARIABLE



CULLING



SPECIALIZATION



DENOISING



TRANSMISSION



And that's it for variable, onto culling.

RAYTRACED SHADOWS

CULLING

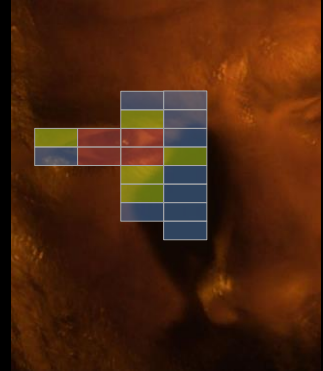


- Working in tiles
- Depth culling
- Static cache culling
- Surface to light cone culling

Consisting of tiles, depth culling, static cache culling, and cone culling.

RAYTRACED SHADOWS

- Tiled classification approach
 - Each light is allocated a set amount of tiles
 - Allocated on conservative CPU estimation of influence
 - Typical classification output
 - Job Queue
 - Indirect Parameters / Signature
- Oversampling limited to tile dimensions
 - Where we trace is a matter of reduction



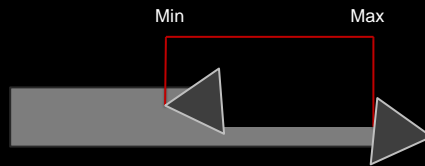
We follow a tiled classification approach, where each light is allocated a conservative set of tiles,

and then produces a typical classification output of a queue and the indirect parameters.

The advantage to this is that oversampling is limited to the tile dimensions, from which it becomes a matter of reduction.

RAYTRACED SHADOWS

- Depth bounds testing, a classic
- Lane samples the pixel depth
 - Reduce min / max over the tile area
 - Reject the lights tile if it does not intersect said bound



So, the first reduction, a classic, reduce the tiles based on the depth bounds.

We reduce the min / max over a tile area, and then reject it if the light doesn't intersect. Simple stuff.

RAYTRACED SHADOWS

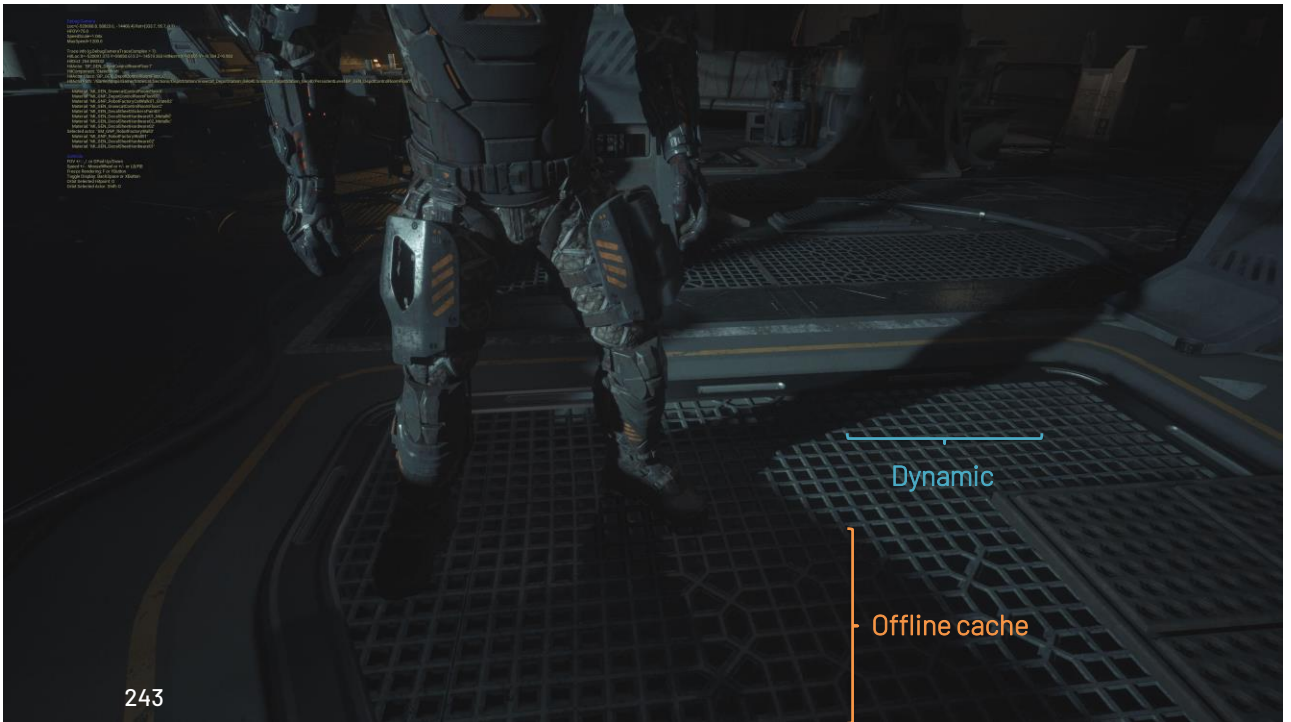


G-BUFFER



DEPTH CULLED TILES

Left, in game render. Right, what the tiled culler sees. Brighter tiles imply lots of lights, darker tiles imply very few.

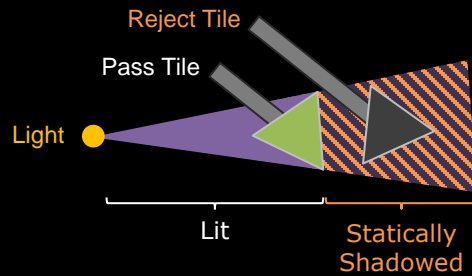


Going back to the stationary caching, we can perform a very useful optimization.

The shadow on the bottom comes from the offline cache, the shadow on the middle right is purely runtime.

RAYTRACED SHADOWS

- Dynamic shadowing only relevant on known lit areas
 - **Reject** tile if the offline cache is fully **statically shadowed**
 - Fast **static occluder** testing



What we can do is use the offline cache for fast occluder checking.

If we have two tiles, first area being known lit, but second area known statically shadowed, we can simply reject the tile as there is no possible dynamic contribution.

RAYTRACED SHADOWS

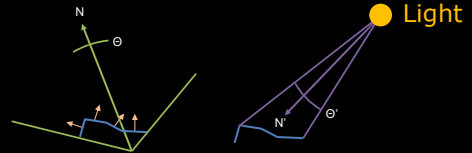


Left, without this occluder checking, right, with occluder checking.

Please note the darkening in the denoted areas.

RAYTRACED SHADOWS

- Cone culling
 - Reject tiles where the summarized $N \cdot L$ has no effect
 - Two cones
 - Tile surface normals
 - Tile surface to light
 - Reject if cone intersection failed
 - Wave intrinsics can perform summarization in a single iteration



And then finally, cone culling.

We can reject the tile entirely if the summarized $N \cdot L$ has no effect.

To do this we summarize two cones, the tiles surface normals, and the tiles surface to light. If the intersection between the two cones fail, we reject the tile.

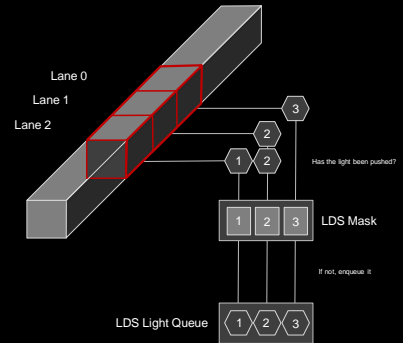
RAYTRACED SHADOWS



Left, without, right, with cone culling. Please see the darkening in the denoted areas.

RAYTRACED SHADOWS

- Reuse the forward rendering light grid
 - Three dimensional, depth slices
 - Collapse all lights within the depth bounds for tile
- One workgroup per tile
 - Parallel loop over active tile range
 - Collect all visible lights
 - Multiple cells may have the same light
 - Interlocked check to see if it's pushed



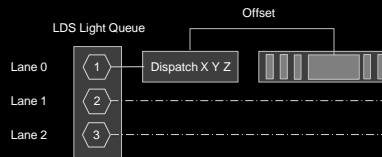
An interesting bit is that we reuse the forward rendering light grid for culling.

This allows us to just collapse all the lights within the depth bounds, without having to perform expensive shape testing all over again.

Initially, more on that later, this was one workgroup per tile. Each carefully reducing the light set over the tile set.

RAYTRACED SHADOWS

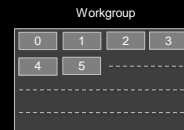
- Standard indirect parameters
- If a light passes the rejection features
 - Allocate a new tile
 - Append some offsets



From here we produce a standard set of indirect parameters, and the associated tile offsets.

RAYTRACED SHADOWS

- Aligning the tile size to raygen (platform) workgroup size is not enough
 - Highly irregular cache signature
 - Tile size has to be exactly the workgroup size
- Classification slowdown, 10x
 - High resource requirements per workgroup
 - Process multiple tiles within a workgroup
 - Light set mask stored externally
 - Each tile allocated a bit



However, we found that just aligning the tile size, for example 32 by 32, to the platform wise ray generation workgroup size was not enough.

A full dispatch was not matching the eqv. indirect dispatch. To match this, the tile size has to be exactly the workgroup size, 8 by 4.

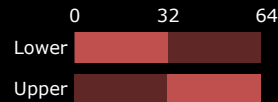
Great, however, classification had a slowdown of 10x, not great.

This is because each workgroup has large resource requirements, the smaller the workgroup size the less real work we have going on.

So, just process multiple tiles in a single workgroup, specifically 32 of them at a time. That gets us close.

RAYTRACED SHADOWS

- Accelerate with wave intrinsics? Waves span across tiles
 - Reorder scheduling so each wave are aligned to the tile regions
- Wave32 / Wave64?
 - High resource requirements per wave, Wave32 sub-par
 - Two tiles in flight at a time, compute wave ops for both tiles!
 - Mask out other tiles data
 - Intrinsics can set lower lane bound



However, if we're processing multiple tiles at a time. It becomes a bit tedious to accelerate things with wave intrinsics as they can span across tiles.

To combat this we just reorder the scheduling so each wave is aligned to the tile regions.

Our tiles are 8 by 4, so wave32 right? No, again, the high resource requirements gets in our way. We need to execute on wave64.

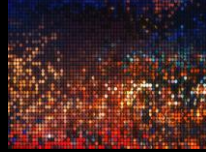
To do this we just process two tiles at the same time, and compute the wave operations for both tiles at the same time. The upper intrinsic masks out the lower band, the lower intrinsic masks out the upper band. Really simple, and really fast.

RAYTRACED SHADOWS

CACHING



VARIABLE



CULLING



SPECIALIZATION



DENOISING



TRANSMISSION



That's it for culling, let's specialize the traces.

RAYTRACED SHADOWS

SPECIALIZATION



- Queue selection
- Tile allocation
- Pre-sum

To do this we'll look at queue selections, tile allocation and pre-sums.

RAYTRACED SHADOWS

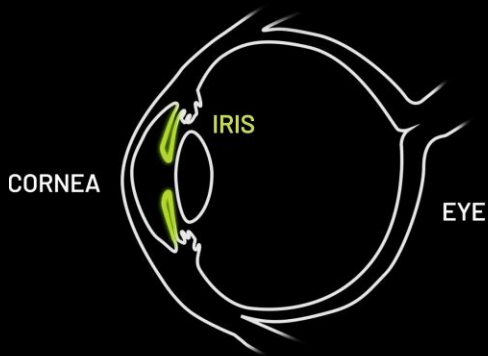
- Raytracing control flow divergence incurs global slowdowns
 - Mechanism to split control flow with localized cost
 - Mechanism to add complex features with localized cost
- We need to **split the pipeline**

The problem is that any control flow divergence,

and additional resource pressure from having the code path at all, incurs a global slowdown.

So, we need a mechanism to split it with localized costs. To do this, we need to split the pipeline.

IRIS DUAL RAYTRACED SHADOWS



RAYTRACING 2 TIMES



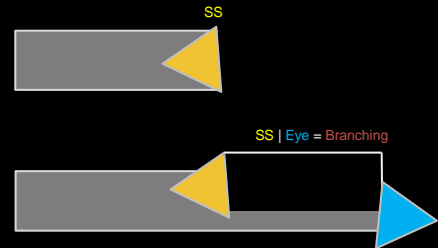
A perfect example of specialization is the iris.

We found that decoupling the diffuse and specular shadowing was vital to sell immersion, as made evident by the video.

But this of course implies two traces per sample.

RAYTRACED SHADOWS

- Improve performance by reducing work per trace
- Shading model feature masks
 - Default
 - **Subsurface Scattering**
 - Eye
 - Branching
- Reduce the feature set as much as possible
- Compile time feature selection



So, we need to reduce the amount of work per trace, we do this by employing compile time feature selection based on the shading model.

So, default, subsurface scattering, eye, and branching if there are multiple models per tile.

RAYTRACED SHADOWS



Let's see it in action.

RAYTRACED SHADOWS

- Depending on the enabled feature set, a light may have multiple queues
 - Each queue selection mask has its own queue
 - Rapid expansion of memory usage

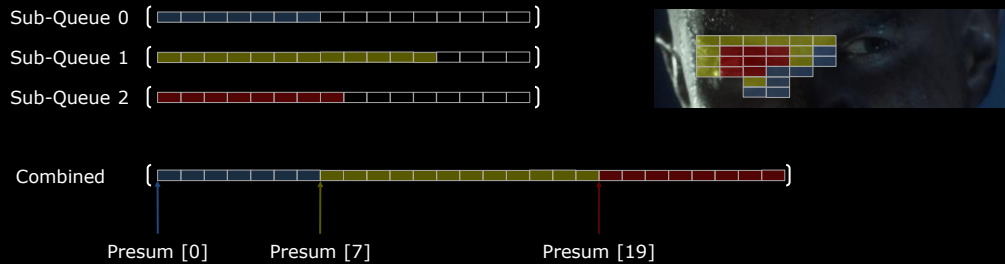


And now, depending on the feature set, a light may have multiple queues.

A naïve implementation would offset the tiles by the queue stride, but this is a rapid expansion of memory usage...

RAYTRACED SHADOWS

- Pre-sum compaction to reduce memory usage
 - Compute the sub-queue offsets for a particular feature
 - Pre-sum pass before the main classifier (light version)



To combat this we run a pre-sum pass that computes the effective intra-queue offsets for a particular feature.

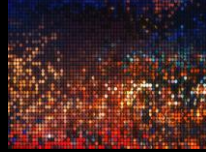
Once the offsets are known, we can merge the queues at runtime, even with the atomic tile allocation, without issue.

RAYTRACED SHADOWS

CACHING



VARIABLE



CULLING



SPECIALIZATION



DENOISING



TRANSMISSION



That was specialization, and now a few notes on denoising.

RAYTRACED SHADOWS

- Vanilla denoiser operates over the lights scissor
 - Only denoise shadowing tiles, skip culled tiles
 - Only denoise tiles with a sufficient variable rate
 - i.e. Ignore tiles with a low shadowing variance
- Numerous optimizations
 - Platform specific optimizations
 - Reduced bandwidth and memory pressure

The inbuilt denoiser works on the lights scissor area, what we do instead of work on the shadowing tiles, and implicitly skip those that have been culled.

Additionally, we only denoise tiles with a sufficient variable rate, i.e. those with a low variance.

This, was the greatest speed up, and what allowed us to denoise our shadows. Behind the scenes we have numerous operations, some platform specific some not, passes on memory pressure, and so on.

They definitely played a part, but are not as interesting to talk about.

RAYTRACED SHADOWS

CACHING



VARIABLE



CULLING



SPECIALIZATION



DENOISING



TRANSMISSION



And finally, transmission.

RAYTRACED SHADOWS

- What do we want?
 - Transmission on **all lights** for consistency
 - Matching Photographic reference
- [Jimenez 2010, "Real-Time Realistic Skin Translucency"]
 - Thickness estimation
 - Transmission estimation

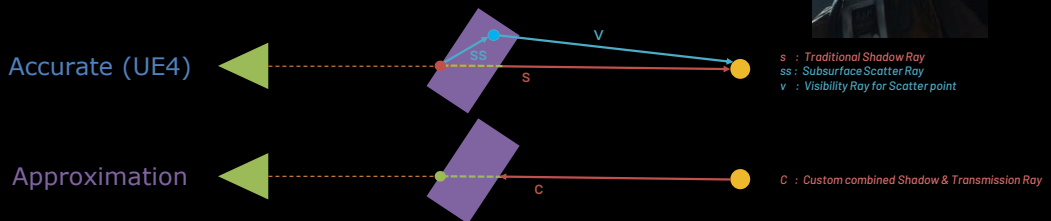


So, what do we want. As usual, everything, so transmission on all lighting, and matching our photographic references.

To realize this we base our transmission model on the 2010 real-time realistic skin translucency, which provides thickness and transmission estimates.

RAYTRACED SHADOWS

- Inbuilt implementation casts 3 rays
 - + An optional early rejection ray
- Reverse the ray for "free" thickness estimation
 - No longer stops at first hit, more expensive
 - Light to surface for thickness, an approximation



(Developed with Miguel Rodríguez)

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023

To inbuilt implementation may cast up to three additional rays for accuracy.

To match our performance budgets, we reverse this ray for free thickness estimation. It no longer stops at the first hit, so the one trace itself is more expensive, but we get away with a single one.

RAYTRACED SHADOWS

CACHING



VARIABLE



CULLING



SPECIALIZATION



DENOISING



TRANSMISSION



So, now that we have all of the infinity stones, how are we looking.

RAYTRACED SHADOWS

- Average performance improvement within an **order of magnitude**
 - ~10x per pass!
 - Not including general ray optimizations

| | | | | | |
|------------|-----------------|-------|-----------|-----------------|-------|
| OFF | [TOTAL] | 61.46 | ON | [TOTAL] | 27.45 |
| | RayTraceShadows | 25.27 | | RayTraceShadows | 3.78 |
| | ShadowsDenoiser | 16.65 | | ShadowsDenoiser | 1.70 |

-36.74ms Saved

-85% Tracing Cost

-89% Denoising Cost

What we get is a speedup of an order of magnitude, roughly 10 times per pass.

And please keep in mind that this does not include general optimizations to our rays, only what we could cleanly, and fairly, toggle.

In this particular example, indicative of proportional savings across the game, we save 36 milliseconds, remove 85% of the tracing cost, and remove 89% of the denoising cost.

This is what allowed us to ship raytraced shadows.

RAYTRACED SHADOWS

- How does it compare to rasterized?
 - Culling & Denoising overhead included
 - Not a fair comparison, visual quality **exceeds** rasterized

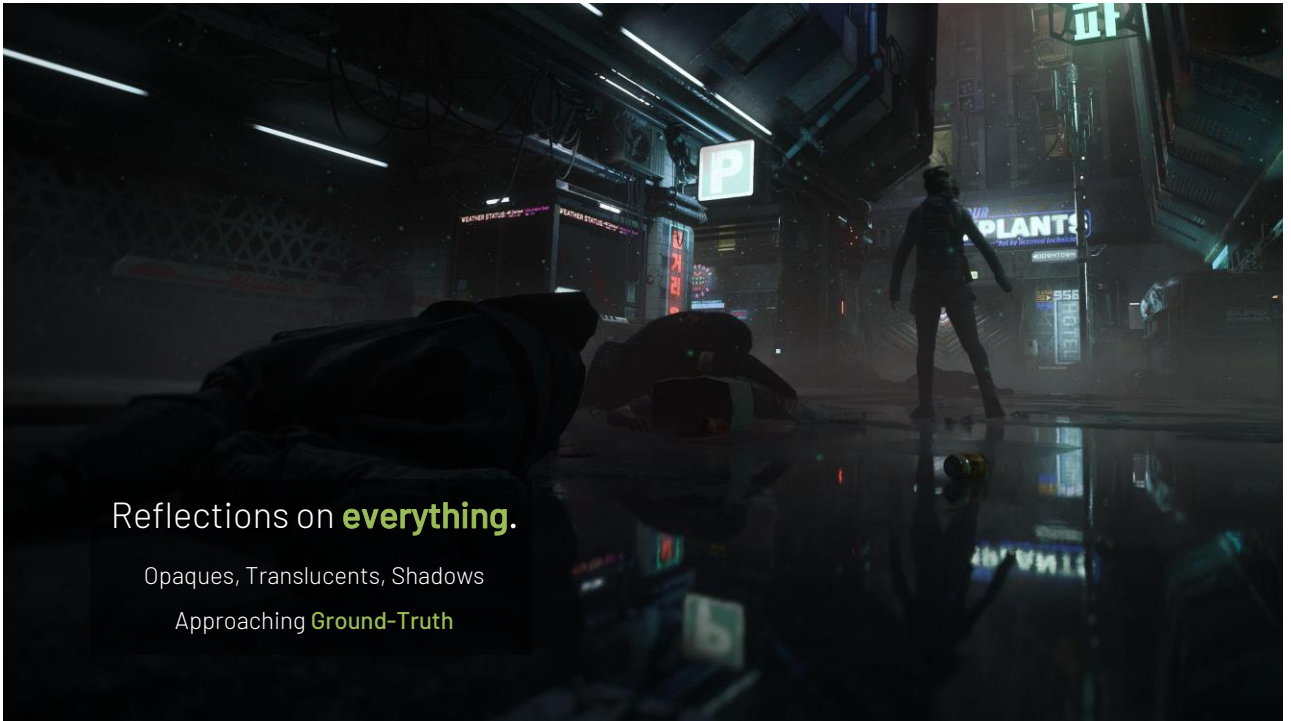
| SCENARIO | RASTERIZED | RAYTRACED | PERCENTAGE |
|--------------|------------|-----------|------------|
| Europa 01 | 5.35 | +2.5 | 67% |
| Tunnels 100 | 3.57 | +4.3 | 45% |
| Minetown 01 | 5.2 | +2.3 | 69% |
| Minetown 150 | 2.8 | +1.7 | 62% |
| Escape 02 | 3.2 | +1.8 | 64% |
| Escape 37 | 7.5 | 0 | 100% |
| Escape 71 | 9.19 | +1.17 | 88% |
| Habitat 33 | 5.7 | +4.5 | 55% |
| Habitat 21 | 6.58 | +2.9 | 69% |
| Habitat 10 | 2 | +2.4 | 45% |

And how does it compare to rasterized? Not that bad, there are certainly cases where rasterized is far ahead, but we're pretty close.

And, please keep in mind that this is an unfair comparison to raytracing, as the raytraced result vastly exceeds the rasterized result.

RAYTRACED REFLECTIONS

So, that was shadowing, what about reflections?



Reflections on **everything**.

Opaques, Transluents, Shadows

Approaching **Ground-Truth**

As with shadowing, the goal is pretty much the same thing. We want reflections on everything, opaques, transluents, and with shadowing.

We want to approach ground truth.

RAYTRACED REFLECTIONS

SHADOWS



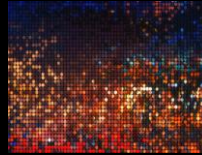
TRANSLUCENTS



LIGHTING REUSE



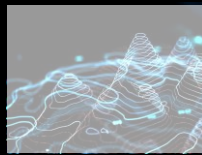
VARIABLE



INLINE TRACE



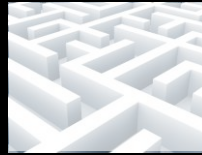
HOLOGRAMS



TEMPORAL



RAY DISTANCES



(HOLOGRAMS and RAY DISTANCES in hidden slides)

So, let's start from the beginning. Shadows.

RAYTRACED REFLECTIONS

SHADOWS



- Offline cache
- Pipelining

First, the kinds of shadowing, then the offline caching, and pipelining.

RAYTRACED REFLECTIONS

- What has shadowing?
 - Static geometry, *offline*.
 - Movable geometry, *runtime*.
 - Movable lights, *runtime*.
- What remains unshadowed?
 - Movable on static geometry. A known sin

We shadow static geometry, movable geometry, even with movable lighting such as the flash light.

However, one thing remained unshadowed, and that is movable on static geometry. The good news is that it's difficult to spot error at runtime, but it is a sin.



What you are looking at here is a mirror reflection, first unshadowed.



And then with shadows.

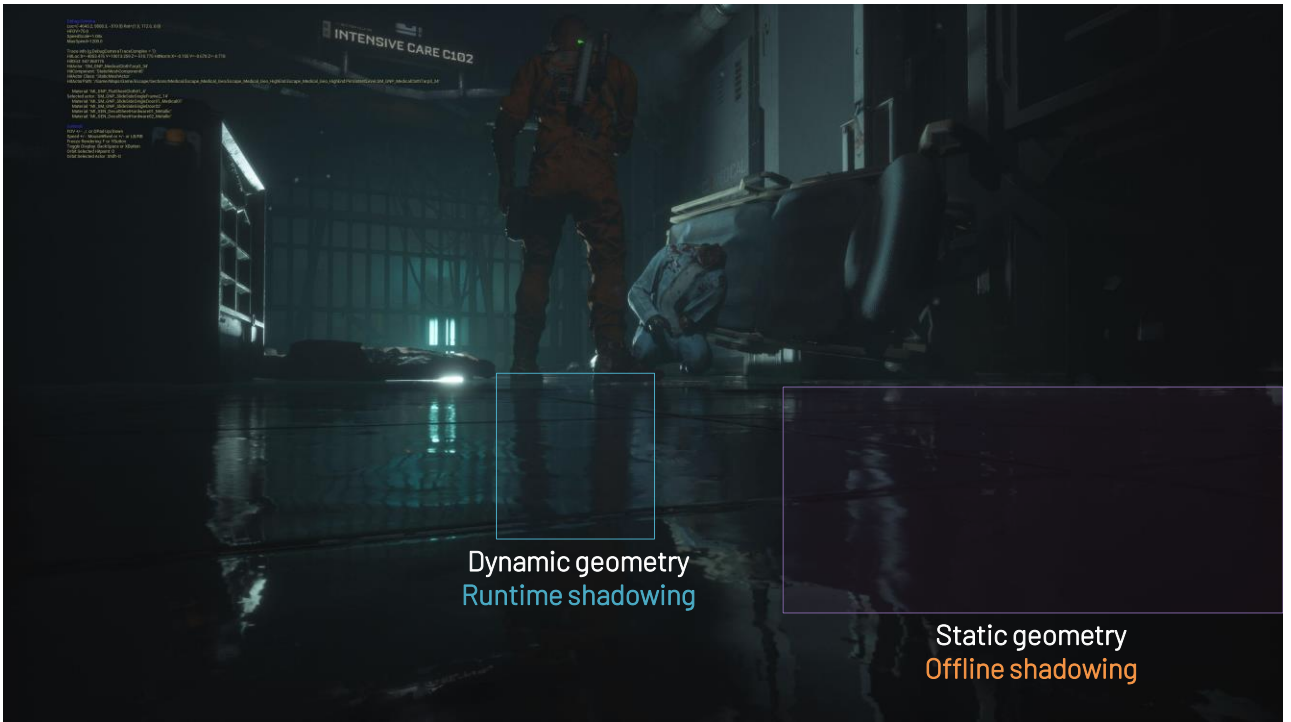
RAYTRACED REFLECTIONS

- Shadows in reflections visually significant
 - Shadow traces exceedingly expensive
- Offline raytracing cache
 - Majority of shadowing already computed by **lightmass**
 - "Free" (negligible) stationary on static shadowing!

So, it's visually significant, but traces are exceedingly expensive.

The best thing we can do is reuse the offline cache as before.

The absolute majority of shadowing is already computed by lightmass, so we can get free stationary on static shadowing.



On the right hand side all the shadowing is a simple texture fetch, extremely cheap.

However, for movable geometry such as the player in the center, shadowing remains a runtime trace.

RAYTRACED REFLECTIONS

- Pipelining is key to performance
- Existing "experimental deferred" path of interest, three step process
 - Initial forced-opaque gather, exports hit information
 - Sort hits on material ids
 - Shade hit positions, with embedded shadow traces
 - Re-traverses if any-hit is needed (i.e. Non-opaque)



But even with that, we need to make it faster. To do that, we need to pipeline things.

Unreal offers a so-call "experimental deferred" path, from which we base our implementation on. This is a three step process.

An initial gather, force-opaque so it stops at the first ray. Sorting on the gather hits based on material ids. Then shading in a manner that promotes material coherence. The last part both re-traces if we hit a non-opaque, and performs embedded shadow traces. So, it's expensive.

RAYTRACED REFLECTIONS

- Move all remaining shadow traces to a packed separate pass
 - Static objects use **offline cache**, dynamic objects perform runtime **traces**
 - Exports packed shadowing, read back during shading
 - Reduced resource usage of benefit to async compute



To combat this we move all the traces to a separate pass, static interactions use offline caching, runtime traces.

Then, all the shadowing data is packed and fed to the shading pass.

This greatly helps the scheduler, and the reduced resource usage is of benefit to async compute and related.

RAYTRACED REFLECTIONS

SHADOWS



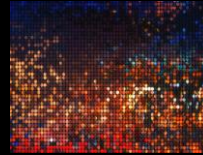
TRANSLUCENTS



LIGHTING REUSE



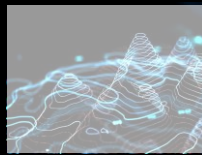
VARIABLE



INLINE TRACE



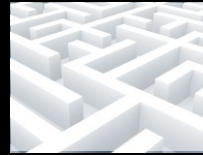
HOLOGRAMS



TEMPORAL



RAY DISTANCES



(HOLOGRAMS and RAY DISTANCES in hidden slides)

That's the shadows, what about translucents?

RAYTRACED REFLECTIONS

TRANSLUCENTS



- Performance
- T-Buffer
- Heuristics
- Future (see hidden slides)

To attack translucents we need to talk about performance, the so-call T-buffer, heuristics, and what lies ahead.

RAYTRACED REFLECTIONS

- Translucent reflections
 - Mirrors, glass, eyes, etc...
- Inbuilt solution casts additional rays, potentially per translucent surface
 - Too expensive
 - **Barely in budget**, how do we get "free" translucent reflections?
 - Can't cast any more rays

When we talk about transluents we're referring to things such as mirrors, glass surfaces, and eyes.

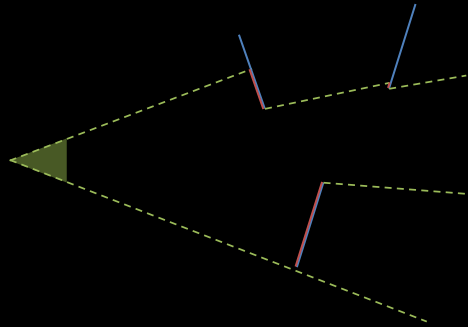
So, a potentially reflective surface above opaque geometry.

The inbuilt solution casts additional rays, potentially per layer, but we are barely in budget.

What do we want? Free translucent reflections, but we can't cast any more rays.

RAYTRACED REFLECTIONS

- Introducing the TBuffer
 - Translucent buffer
 - Geometric pass, rendering translucent geometry data
 - Depth tested
- Only interested in top-most surface



Introducing the T-Buffer, T being transluents. It's essentially just a geometry pass that renders the front-most translucent geometry.



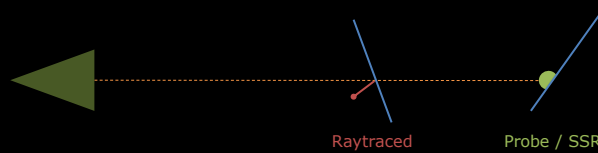
On the left hand side you may find reflections where the primary ray is derived from the T-Buffer, in this case a glass pane.

On the bottom you may find reflections where the primary ray is derived from the G-Buffer. All opaque.

Quite a cool effect if you ask me.

RAYTRACED REFLECTIONS

- Reflection ray uses TBuffer instead of GBuffer if closer
- Top most layer uses **raytracing**
- Subsequent layers use **probe lookups and SSR**



Implementation wise, to avoid casting more rays, the reflection ray simply uses the TBuffer instead of GBuffer if it's closer.

Given that this may result in multiple layers, last being the GBuffer, the front most layer always raytraces, and all successive layers either use probe fallbacks or SSR, depending on which has best visibility.

RAYTRACED REFLECTIONS

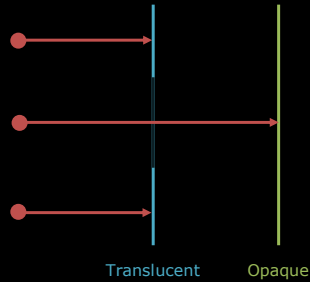
- We also tried additional various layer heuristics
 - Roughness assumptions
 - Reflective estimation, f.x. EnvBRDF evaluation
 - ...
- All **error prone**, fail under certain conditions, a mess

Something I should note is that we tried many heuristics for where to trace, such as making assumptions on the roughness, environment brdf, etc..

However, we found that it was too error prone, and whenever we thought we had something, it broke somewhere else. Frankly, a mess, simple is best.

RAYTRACED REFLECTIONS

- Future, explore temporal solutions
 - Checkerboarding on layers
 - Temporally converge samples on layer with greatest contribution



Now, in the future what I believe makes most sense is something akin to checkerboarding on layering. Then converging the samples on the layer with the greatest contribution temporally.

RAYTRACED REFLECTIONS

SHADOWS



TRANSLUCENTS



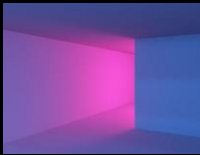
LIGHTING REUSE



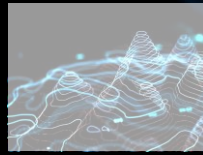
VARIABLE



INLINE TRACE



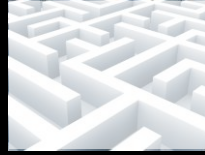
HOLOGRAMS



TEMPORAL



RAY DISTANCES



(HOLOGRAMS and RAY DISTANCES in hidden slides)

That was translucents, let's talk about reusing lighting data.

RAYTRACED REFLECTIONS

REUSE LIGHTING



- Reusing lighting data
- Rasterization parity
- Volumetrics

Which will cover the refuse, rasterization parity, and volumetrics.

RAYTRACED REFLECTIONS

- Always raytrace, never relies on screen space traces
 - Ground truth raytracing
 - However, reuse lighting from previous frame, if available
- Our BVH is close to parity with rasterization, but misses a few things
 - Tiny Niagara particles (millions), certain decals
 - Reusing screen data helps this

However, when I say reusing screen data, it is not in the hybrid sense. We are always raytracing, we never rely on screen space traces. Stay ground truth!

What I mean is that we reuse lighting data from the previous frame, if available.

Why? First, performance, second, and more importantly. Our BVH is very close to parity with rasterization, however, it may miss a few things.

Namely large niagara particle systems and certain decals. This is where reusing screen data helps.



This is with a BVH only trace



This is with reusing screen data.

[Toggle back and forth]

As you can see, it really helps to bring back some of the missing things.

RAYTRACED REFLECTIONS

- Approaching ground truth requires parity for consistency
 - Light functions
 - IES profiles
 - Volumetrics
 - Etc.

So, on this pursuit we had a large push on parity.

Lighting functions need to be the exact same between rasterization and raytracing, same goes for IES profiles, volumetrics, etc...

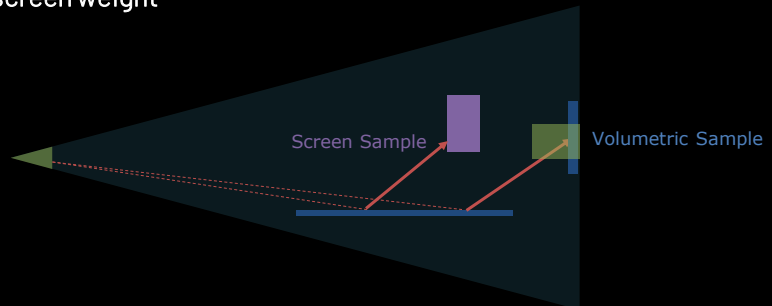
But the volumetrics were a large issue.



Without volumetric contribution, we unfortunately have a massive visual disconnect. And not a pretty one.

RAYTRACED REFLECTIONS

- Screen sample includes volumetric data, raytracing does not
- Unreal uses a froxel grid for volumetric integration
 - Sample volumetric data on occluded hits
 - Careful with double contribution, blend inversely with screen weight



This became a problem when we started to reuse screen lighting data, as it includes volumetrics while raytracing does not.

Saving point is that unreal uses a froxel grid to represent volumetric integration, which exceeds the rasterized first hit.

So, in raytracing, we sample said data if we could not sample the previous screen data. Please be very careful with double contribution, it'll bite you.

And a small note, I think for the future we either go with expensive integration, or experiment with pre-integrated spaces, my money's on the latter.



Without sampling on raytracing.



With sampling on raytracing.

[Toggle a few times]

Really helps ground the effect.

RAYTRACED REFLECTIONS

SHADOWS



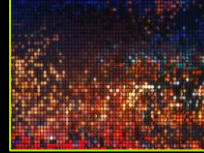
TRANSLUCENTS



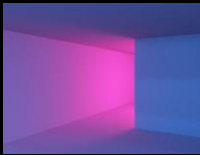
LIGHTING REUSE



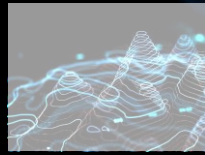
VARIABLE



INLINE TRACE



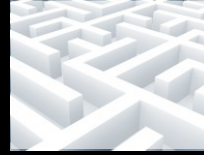
HOLOGRAMS



TEMPORAL



RAY DISTANCES



(HOLOGRAMS and RAY DISTANCES in hidden slides)

That was lighting, let's talk about variable rate tracing.

RAYTRACED REFLECTIONS

VARIABLE



- Working in tiles
- Variable rate scheduling
- Perceptive importance
- Variable resolution (see hidden slides)

Which will include tiles, variable rates, perceptive important, and variable resolution.

RAYTRACED REFLECTIONS

- Too slow, and too many rays where it doesn't matter
 - Similar problem space to raytraced shadows
- Reintroduce **tiles**
 - Same scheduling paradigm
 - Far simpler problem space than raytraced shadows
 - Earlier rejection of rays

The problem is similar to shadows, it's still too slow, and we're shooting too many rays where it doesn't matter.

So, let's re-introduce tiles. It's a far simpler problem space than raytraced shadows, from it we get earlier rejection of rays.

Given that I already went over the concepts during shadowing, I won't go into too much detail here.

RAYTRACED REFLECTIONS

- Variable rate tracing
 - Extends [Ray-Traced Reflections in 'Battlefield V', GDC 2019]
 - Used roughness as variable factor
 - Roughness not necessarily indicative of final contribution
 - Variable rates instead computed from reflection variance
- Calculate variance across tile
 - One frame delay, as with raytraced shadows
 - Carefully reprojected

And, let's add variable rate tracing.

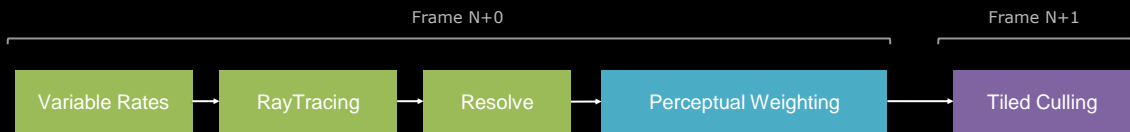
This is actually extending the work done in the 2019 Ray-Traced Reflections in Battlefield V, which uses roughness as the variable factor.

However, we believe that roughness is not necessarily indicative of the final contribution, so we instead compute the rates from the reflection variance across the tiles.

Again, a one frame delay, so carefully reproject things.

RAYTRACED REFLECTIONS

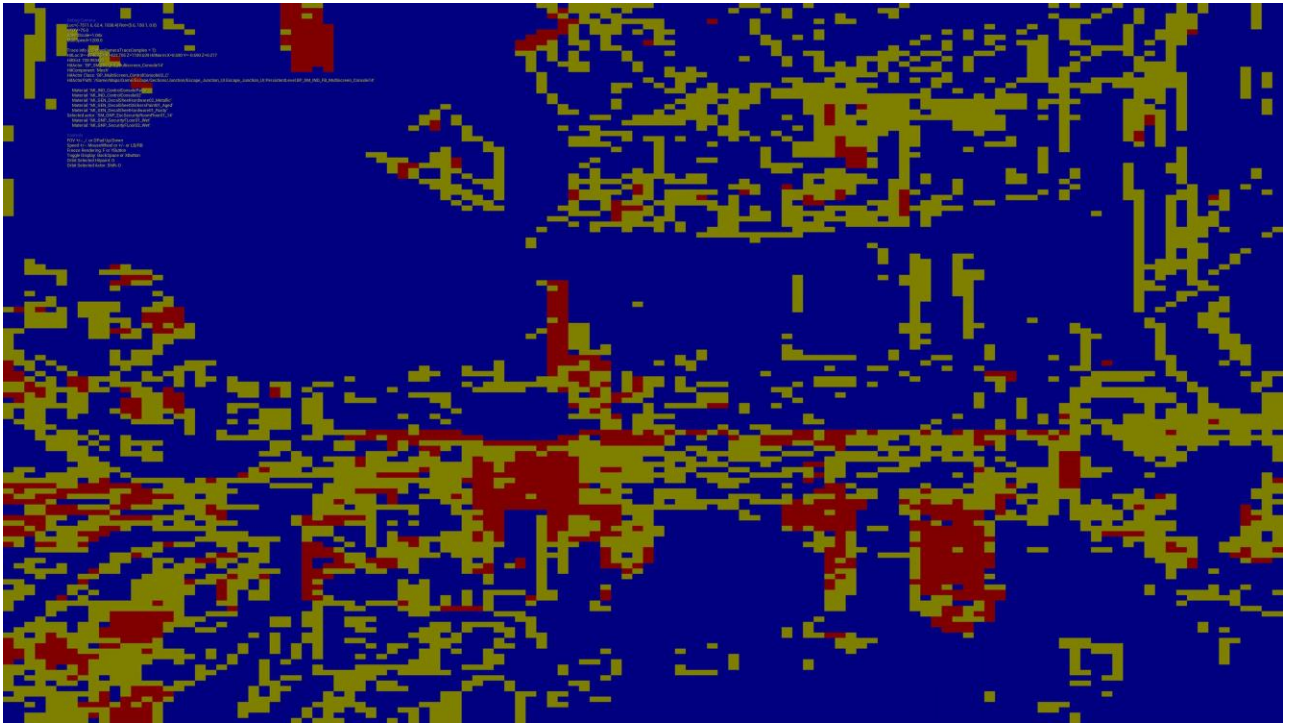
- **Perceptive importance** equally relevant
 - Compare linear reflection results with linear final scene values
 - Compute variance on weighted samples



Let's bring over the perceptive importance as well, we compare the linear reflection results with the linear final scene values. From this we compute the variance on the weighted samples.



And showing all of those things in action, please note the mirror-like reflections on the floor below.



And these are the resulting variable rates computed.

[Toggle it a few times]

Note how the samples are concentrated on areas with reflections.

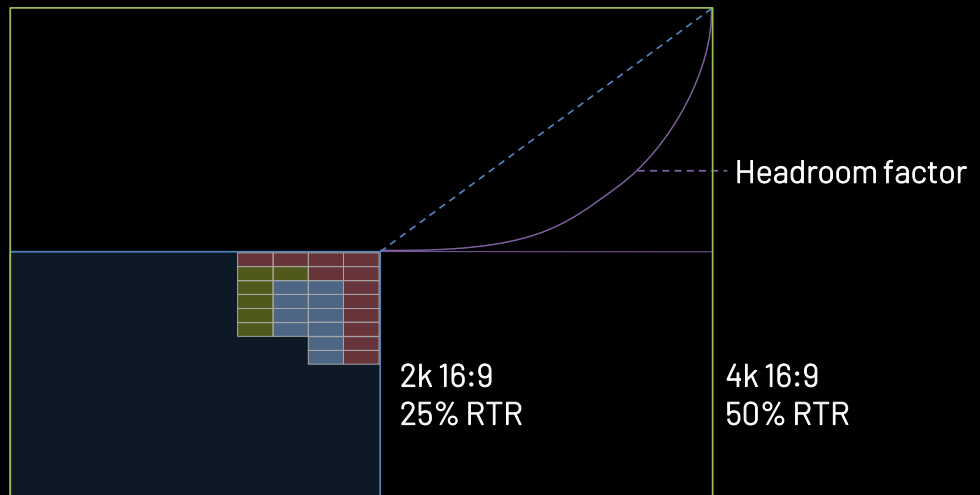
RAYTRACED REFLECTIONS

- Reflection resolution is variable between 50% - 25% (both axes)
 - Variable rates and variable resolutions? What?
 - Variable resolution came first
 - While questionable, it is very scalable
- Variable resolution computed from performance headroom
 - Headroom factor expressed as scaled exponential of the frame dynamic resolution
 - At target frame resolution 2k 16:9, 25%
 - At native frame resolution 4k 16:9, 50%
 - Quite a few sequences hit 4k

In addition to variable rates, we also have general variable resolution. While this is an odd combination, it's simply because one came before the other, namely variable resolution.

We decided to keep this as it proved very scalable. The general reflection resolution varies from 50% to 25%, where the actual percentage is derived from the performance headroom from target, 2k 16:9, to native 4k. Scaled exponentially to give room in the lower bounds.

RAYTRACED REFLECTIONS



Just to visualize things, 2k 16:9 being the target, all the way to native 4k. Our headroom factor proved most useful as an exponential between the two.

RAYTRACED REFLECTIONS

SHADOWS



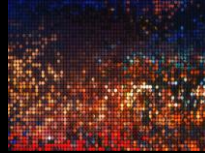
TRANSLUCENTS



LIGHTING REUSE



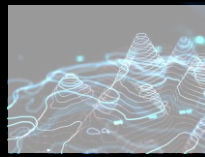
VARIABLE



INLINE TRACE



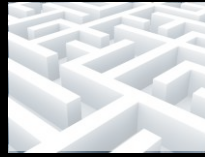
HOLOGRAMS



TEMPORAL



RAY DISTANCES



(HOLOGRAMS and RAY DISTANCES in hidden slides)

With variable rate tracing, let's move on to inline tracing.

RAYTRACED REFLECTIONS

INLINE TRACE



- Any hits
- Inline summarization

And specific to the problem of any hits, and what we call inline summarization.

RAYTRACED REFLECTIONS

- Better! But still not good enough.
 - Any hits incredibly (proportionally) expensive
 - Many assets over reliant on opacity mapping
 - Too late to address, somewhat of an unfortunate situation
 - We need to reduce any-hit evaluations, key is unordered traversal



The problem is that our any hit stages are proportionally incredibly expensive, with too many assets over reliant on opacity mapping.

It was unfortunately too late to address through content, so we had to look elsewhere.

To reduce any hit evaluations, we need to attack the unordered traversal.

RAYTRACED REFLECTIONS

- Extend gather pass to perform “inline summarization”
- Lightweight inline trace that summarizes the effective ranges
 - Final opaque hit position
 - Effective any-hit ranges
 - Reduced by final hit



What we do is extend the gather pass, first one, to perform inline summarization, which essentially determines the effective ranges for any-hits and the final opaque.

And most importantly, reduces the any-hits by the final opaque and other metadata that could indicate termination.

RAYTRACED REFLECTIONS

- Schedule additional pass Gather Complex if any-hit is needed
 - i.e. There is a real any-hit before the final opaque
 - Export final opaque hit point
- Shading pass no longer performs traversal
 - Only evaluates hit positions / materials (vendor extensions)

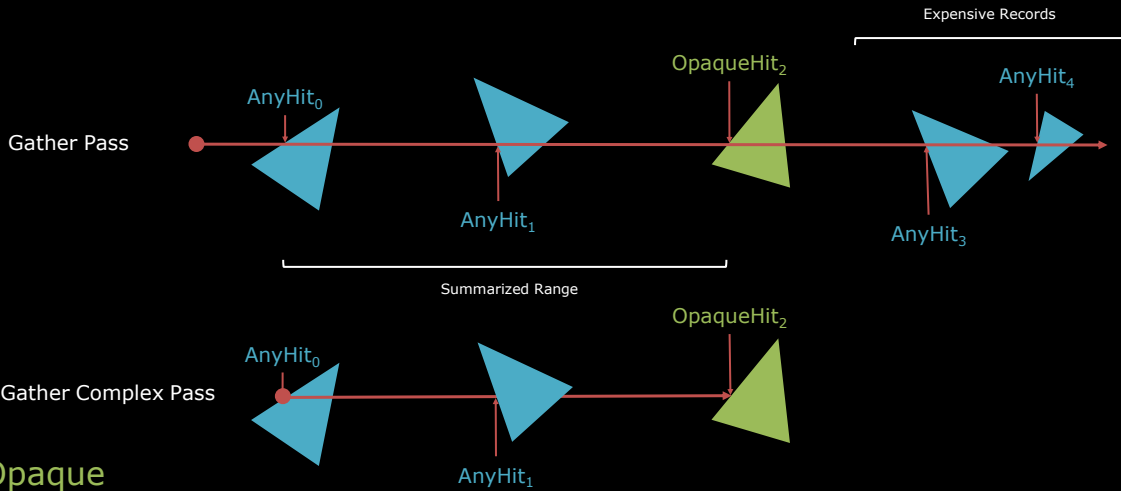


Then, based on this reduced range, instead of performing additional re-traversal during shading, we schedule a dedicated pass.

This pass only runs if there is any potential any-hits to evaluate. Once done, it exports the final hit point.

This not only improves the re-traversal based on the reduced ranges, but also general shading performance due to the reduced resource usage and better material coherency on shading, as the sorting pass is now sorting the real final hits.

RAYTRACED REFLECTIONS



Opaque
Non-Opaque

That was a lot of text, let's visualize it.

First ray, the gather pass, goes through all the potential hits.

Now typically the re-traversal would evaluate the expensive records at the end, but this pass is fully inline.

And extremely cheap at that, couple hundred microseconds in addition at most. Now that we know the actual ranges, the gather complex solely operates on this range. And does not evaluate the expensive records at the end.

RAYTRACED REFLECTIONS

- Great yields
 - Up to 3ms in worst offenders
 - Average is much lower
- PC drivers uneven gains / losses
 - Disabled

The performance you get back from it depends heavily on the platform and implementation, however, for our use cases this proved great in production. And most importantly, it worked best in cases where we were bottlenecked by reflections.

RAYTRACED REFLECTIONS

SHADOWS



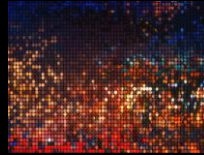
TRANSLUCENTS



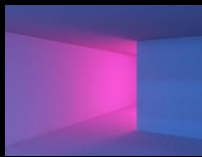
LIGHTING REUSE



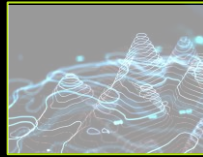
VARIABLE



INLINE TRACE



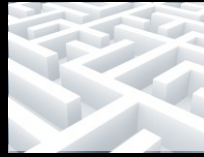
HOLOGRAMS



TEMPORAL



RAY DISTANCES

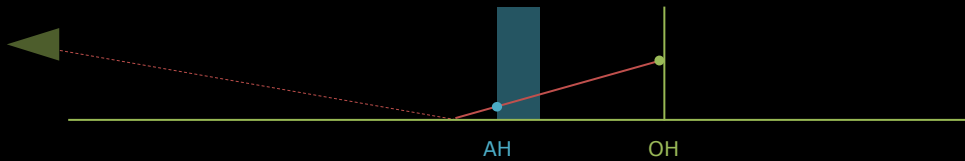


(HOLOGRAMS and RAY DISTANCES in hidden slides)

So, now that we've got a little bit of performance head room, let's talk about holograms.

RAYTRACED REFLECTIONS

- **Shading** picks up emissive radiance on the final hit point
 - **Gather Complex** pass accumulates radiance along the ray
- Ordered or unordered?
 - Kept unordered for performance, *may result in missed occlusion*
 - Inline summarization helps, content rarity made this acceptable



The general shading pass already accounts for emissive contribution, however, not along the ray.

So, let's include it. Whenever the gather complex pass evaluates an any-hit, accumulate the radiance.

With that comes the question, ordered or unordered? We decided to keep it unordered for performance, which may result in missed occlusion. However, inline summarization greatly helps reduce the chances of this, and general content rarity made it acceptable.



And personally, I love the effect.

RAYTRACED REFLECTIONS

SHADOWS



TRANSLUCENTS



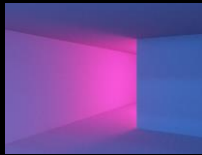
LIGHTING REUSE



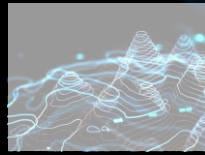
VARIABLE



INLINE TRACE



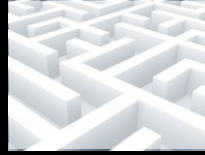
HOLOGRAMS



TEMPORAL



RAY DISTANCES



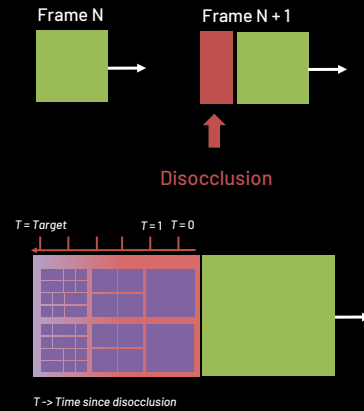
(HOLOGRAMS and RAY DISTANCES in hidden slides)

Let's talk a bit about reflections and temporality.

RAYTRACED REFLECTIONS

- Use IGN [Jimenez 2014] across the board
- Hierarchical Blurring
 - Temporal counter since disocclusion
 - Reduce hierarchical blurring over time
 - Moving average instead of exponential moving average

$$\alpha = 1 - \left(\frac{1}{t + 1}\right)$$



We use IGN noise across the board, as it seemed a better fit for our use cases.

And to deal with disocclusion we simply employ a hierarchical blur, whose size is tied to the last disocclusion time.

The more recent the disocclusion, the greater the blur. We found that to be sufficient.

RAYTRACED REFLECTIONS

SHADOWS



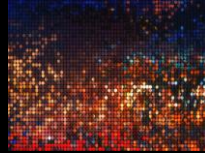
TRANSLUCENTS



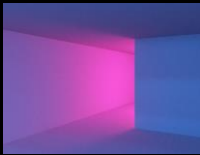
LIGHTING REUSE



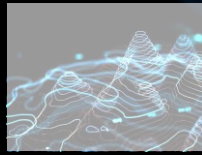
VARIABLE



INLINE TRACE



HOLOGRAMS



TEMPORAL



RAY DISTANCES

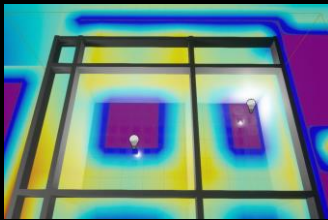


(HOLOGRAMS and RAY DISTANCES in hidden slides)

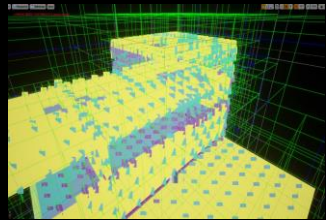
And finally, ray distances.

RAYTRACED REFLECTIONS

- On a given area, precompute the max ray distance
 - Thin corridors, constrained spaces, ideal environments
 - Most effective ray limits are a fraction of the world bounds
- Not shipped, ran out of time



Offline volumetric cache



Offline surfel cache



Runtime progressive spatial-hashing cache

We experimented with precomputing the maximum ray distances across all levels, since most effective ray limits are a tiny fraction of the world bounds. It really is the ideal environment.

We opted not to do this in lightmap space, as the memory requirements are large. And instead evaluated offline volumetric caches, offline surfel caches, and runtime progressive spatial hashing caches.

Unfortunately this did not ship, as we ran out of time in general. However, if I was to place my bets on any of this, it would be the progressive runtime caching.

PLATFORM OPTIMIZATIONS

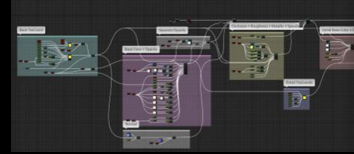
That was reflections. But we're not done, while there's a heap of general optimizations, I want to touch on a few of the most impactful ones.



The things that made us able to ship with this kind of fidelity.

PLATFORM OPTIMIZATIONS

- The material cat and mouse game
 - Wonderful tool for content creators
 - A "challenge" for engineers
- Current raytracing pipelining requires that the per-stage GPR requirements are the worst case GPR requirements.
 - Something to revisit in the future
 - A single material being streamed in can tank performance
 - Why is the closest hit doing **video decoding**?
 - Why is the closest hit doing **texture space tracing**?



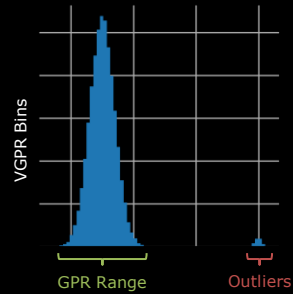
The first problem is known to many, and that's the material cat and mouse game, while it's a wonderful tool for content creators, it can be a challenge for engineers.

This is especially so as raytracing pipelines require the per-stage GPR is that of the worst case.

So, if you suddenly have a single material streamed in with large requirements, performance can tank across the board. We had a fun few cases regarding video case decoding and texture space tracing, and by fun I mean a jira issue.

PLATFORM OPTIMIZATIONS

- Determine the **known-good** material GPR ranges
- Limit VGPR, let the **rest** spill to scratch memory
 - Known good content betters
 - Expensive content worsens



To combat this, we determine the known good GPR ranges, and simply limit it to that.

Any usage beyond that will spill to memory.

What this means in practice is that known good content betters while expensive content worsens, that's fine. We want to address expensive materials separately, hit shaders should be simple shaders.

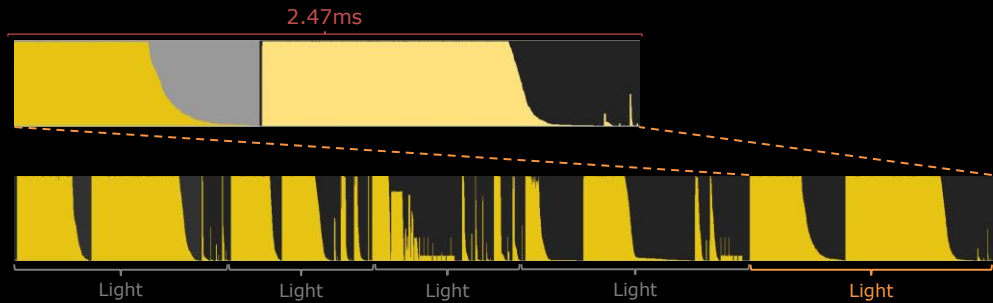
PLATFORM OPTIMIZATIONS

- Split up hit shader tables on the use
 - Raytraced shadows
 - Raytraced reflections
- Shadow testing pipelines do not suffer from VGPR requirements of reflection evaluation. And vice versa.
 - Can be split in a much more fine grained manner
 - Bottlenecked by CPU material parameter binding costs(UE5!)

Additionally, we split up the shader tables on use, so, raytraced reflection hit shaders do not affect raytraced shadows hit shaders, and vice versa.

PLATFORM OPTIMIZATIONS

- Long raytracing shadow tails
 - Unfortunate traversal paths
 - Unfortunate any-hit evaluations (f.x. Hair)



(Captured on PIX-XXS)

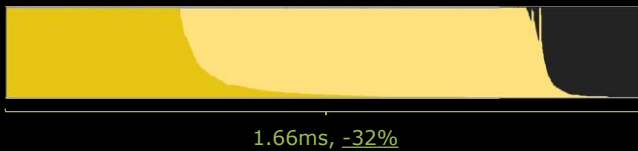
Advances in Real-Time Rendering in Games course, SIGGRAPH 2023

Another issue is tails, caused by unfortunate traversal paths, any-hit evaluations, and anything else particular to only a few rays out of the many.

This is especially bad with numerous traces after each other and pipelining limitations.

PLATFORM OPTIMIZATIONS

- Hide tails and latency with overlapped execution
 - Batch as many raytracing calls together as possible
 - F.x. 32 lights with long tails becomes one effective tail
 - Virtually eliminates the problem
- Overlapped passes must share the same VGPR scratch layout!



(Captured on PIX-XXS)

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023

To combat this, we hide tails and their associated latency with batching.

So, if you have 32 lights, all their tails become one long tail. In this particular light we went from 2.47ms to 1.66ms on XSX, 32% reduction.

This comes at a cost of increased memory, the more batched traces the more memory you need to keep alive at a time. Also, for those messing with scratch spaces, be very careful with the overlapped passes and shared scratch layouts.

PLATFORM OPTIMIZATIONS

- Raytracing reflection tails
 - Overlap? Not possible, inter-stage dependency

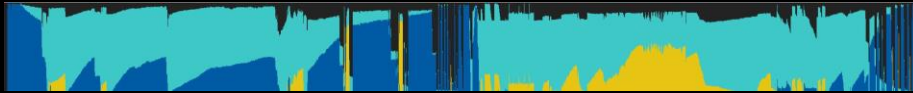


Frame Time 32.5ms

So, for reflections we'd want to do the same, but can't, as each stage depends on the last.

PLATFORM OPTIMIZATIONS

- Async compute to the rescue!
 - Graphics queue has higher priority by default
 - Kickoff and sync points are half a frame apart
 - Reduce allocated wave limit to 50% for async reflections
 - Gain in the milliseconds, on average $\sim 2.5\text{ms}$
- Carefully manage async compute and graphics scratch spaces!



Frame Time 30.15ms (- 2.35ms)

(Captured on PIX-XXS)

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023

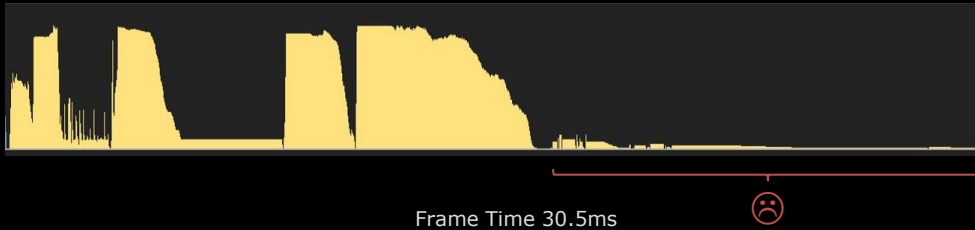
To combat this we offload it to async compute, and specifically reduce the allocated wave limit to 50%.

Our reflection kick off and sync point are half a frame apart, we let the graphics work run as undisturbed as possible, especially given the resource requirements.

The gains are in the milliseconds, in this particular case 2.35ms on XSX.

PLATFORM OPTIMIZATIONS

- Acceleration structure building not free
 - BLAS skinned builds expensive
 - TLAS building not making effective use of hardware



Frame Time 30.5ms



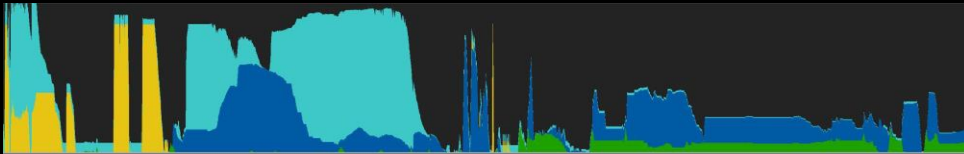
(Captured on PIX-XXS)

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023

Unfortunately, there are similar problems with acceleration structure building, skinning builds are expensive and TLAS building is not making effective use of hardware.

PLATFORM OPTIMIZATIONS

- Schedule on async compute
 - Gains in the milliseconds
 - Even on PC (~1/2 ms)



Frame Time 29.6ms (- 0.9ms, - 80% 😊)

(Captured on PIX-XXS)

Advances in Real-Time Rendering in Games course, SIGGRAPH 2023

So, offload it to async compute as well. In this particular case we save 0.9ms, a reduction of 80% on XXS. Practically free at this point.

PLATFORM OPTIMIZATIONS

- This talk covered the large features
- Thousands of smaller optimizations
 - Not very interesting to talk about
 - Combined contribution had a large effect on performance
- Months of tracking down GPU crashes across platforms
 - Thank you for **vendors** for the extensive help

And that's mostly it. Again, this talk covered the large features, but I must reiterate that we have thousands of smaller optimizations across the board that combined had a large effect on performance. They are just not that interesting to talk about.

And, as usual, the many months spent on tracking down GPU crashes, a big thank you to the vendors here. Thank you!

KEY TAKEAWAYS

So, besides the technical and algorithmic bits, what are we to take away?

KEY TAKEAWAYS

- Algorithmic and platform optimizations are great
 - But doesn't get us all the way
- What landed us in budget is optimizations for our content
 - Any generalization we can make due to content particularities
- Raytracing not far from matching rasterized performance
 - Our belief that we will eventually match and **exceed it**

From my side, algorithmic and platform optimizations are great, but they don't get us all the way.

We need to optimize for the content that we have, and generalization or assumption you can make because of it.

And, it is my belief that we aren't that far from matching rasterized performance, it is my personal opinion that we'll eventually not only match, but exceed it.

OUR VISION

- **Ground Truth**
 - Minimize shortcuts
 - Pure runtime raytracing + Accurate cached lighting
 - Long range light attenuation radius
- **Consistency**
 - Materials to respond correctly in all situations
 - Virtually all lights to have shadows (raytraced)
 - Virtually all surfaces to have accurate reflections (raytraced)
- **Observation**
 - Train to discern subtleties
 - Photo reference
 - Digital Doubles



To summarize our journey, I would like to get back to our vision.

We have shown how we followed the ground truth, avoiding typical real-time shortcuts, and investing in raytracing and accurate lighting.

We have shown how we pursued consistency in the materials through a new expressive BRDF and Realis, and also in lighting by allowing for accurate shadows and reflections during gameplay.

Finally, we shown how we learnt from the real world through observation, and used that as the true ground truth for us to follow.

THANK YOU

Special thanks to **Glen Schofield** & **Natasha Tatarchuk**

Special thanks to **Sony, AMD & Microsoft**

AMD

Microsoft

| | | | | | | |
|-----------------|------------------|-----------------|-------------------|---------------------|----------------|--------------------|
| Andy Yelland | Glauco Longhi | Luke Iwanski | Jonas Gustavsson | Bernat Munoz Garcia | Troy Smith | Pavel Martishevsky |
| Atsushi Seo | Hampus Siversson | Marcin Gollent | Keri Taylor | Tobias Fast | Nico May | Phillip Proffitt |
| Brandon Ehle | Jay Ryness | Maria Gavara | Francois Guthmann | John Hartwig | James Standard | Jack Elliot |
| Chan Sarinyamas | Jonathan Hudgins | Mark James | Pierre-Yves Boers | Ameer Jallil | Adam Miles | Jordan Saunders |
| Chris Stone | John Lee | Nora Falcon | Sebastien Vince | | David Cook | Cole Brooking |
| Demetrius Leal | Jon Robins | Scott Defreitas | | | | |
| Eddy Toomey | Jorge Segura | Stacey Hirata | | | | |
| Edu Sanchez | Kim Libreri | Steve Papoutsis | | | | |

That concludes our talk.

Thank you for your attention.

If we have some time left, we'll be happy to answer any questions that you may have.