

Welcome to Translating Art into Technology: Physically Inspired Shading in Destiny 2

My name is Alexis Haraux, I've been at Bungie since 2012 working as a Graphics Engineer on both the original Destiny and Destiny 2.

Before I go into further technical detail, let's take a look at some Destiny 2 visuals.



Destiny 2 is a fast paced action shooter with a wide variety of environments to explore, familiar, futuristic, alien robot worlds, covered with vegetation.

There are many kinds of enemies to fight for which you have a large collection of armor and weapons that you can earn.

At a higher level, Destiny 2 is magic, sci-fi and fantasy - which we call "Space Magic" embedded in realism.

TRANSLATING WHAT?



BUNGIE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

The talk is entitled translating art into technology, what did that mean to us?

A general approach is to start with a visual goal, which goes through art direction, technical direction, feature implementation, and finally arriving at content production that provides the final visual result.

Going from art concepts to engineering, back to art while preserving and refining the original goal is what we call translation.

AGENDA

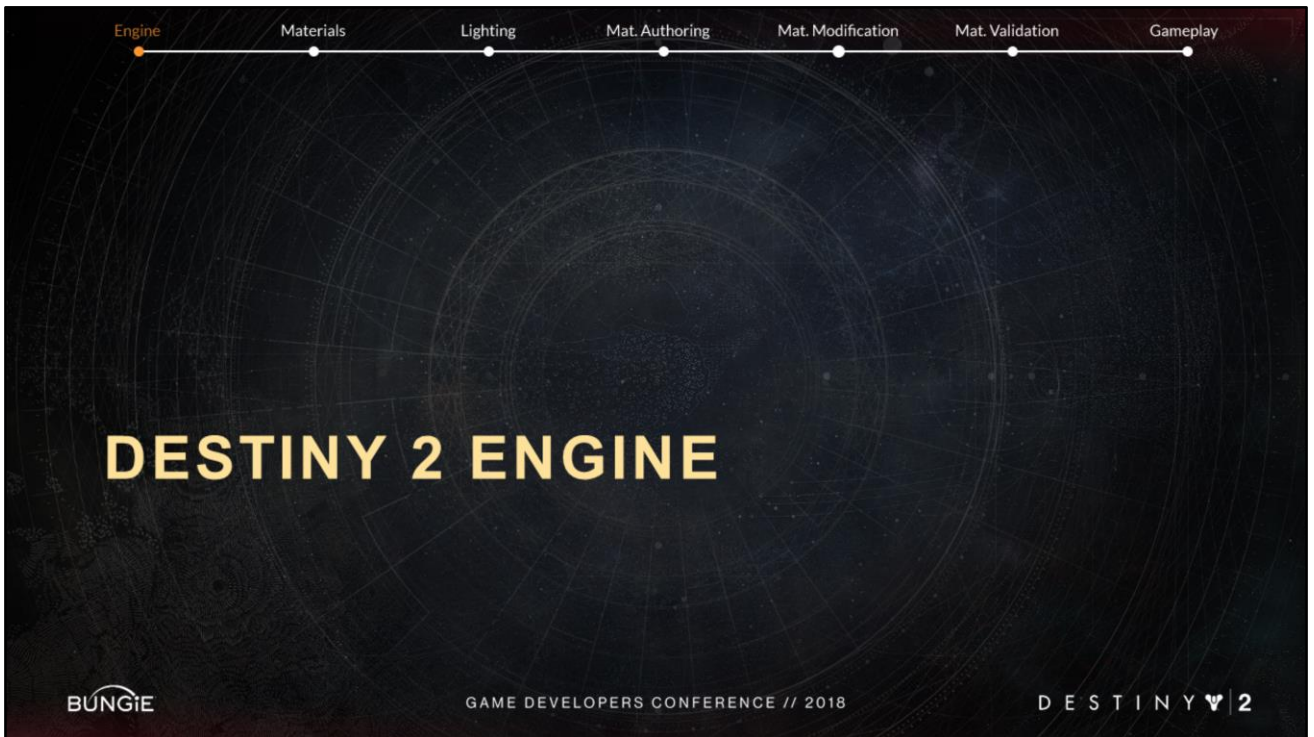
- Destiny 2 Engine
- Materials
- Lighting

- Material Authoring
- Material Modification
- Material Validation
- Gameplay



To explain how we did this, the talk will be split into two parts.

In the first part I will go over the engineering side of the process, starting with a quick overview of our engine and its gameplay requirements, then I will cover the material model and lighting features we added for Destiny 2. After that I will hand it off to Nate who will talk about how we authored, modified, and validated materials, closing with how we further adapted the engine to our gameplay needs.



Let's start with an overview of the Destiny 2 engine.



For some context, let's look at our gameplay and content authoring requirements:

Dynamic time of day

Lots of moving objects

Open World, long view distances and streaming requirements

We author our content as polygon soup: triangles intersect in arbitrary ways

Content is not aligned with any axis or regular grid

Interiors and exteriors not well defined, we like to make buildings with lots of holes and half broken walls

REQUIREMENTS

Dynamism and scale

- Dynamic time of day
- Dynamic objects
- Open World

Authoring

- Polygon soup
- No world/grid alignment
- Interior/exterior

Variety and volume

- Complex shaders
~50k / world
- AI count (50)
- Player count
9 in combat
26 in non-combat

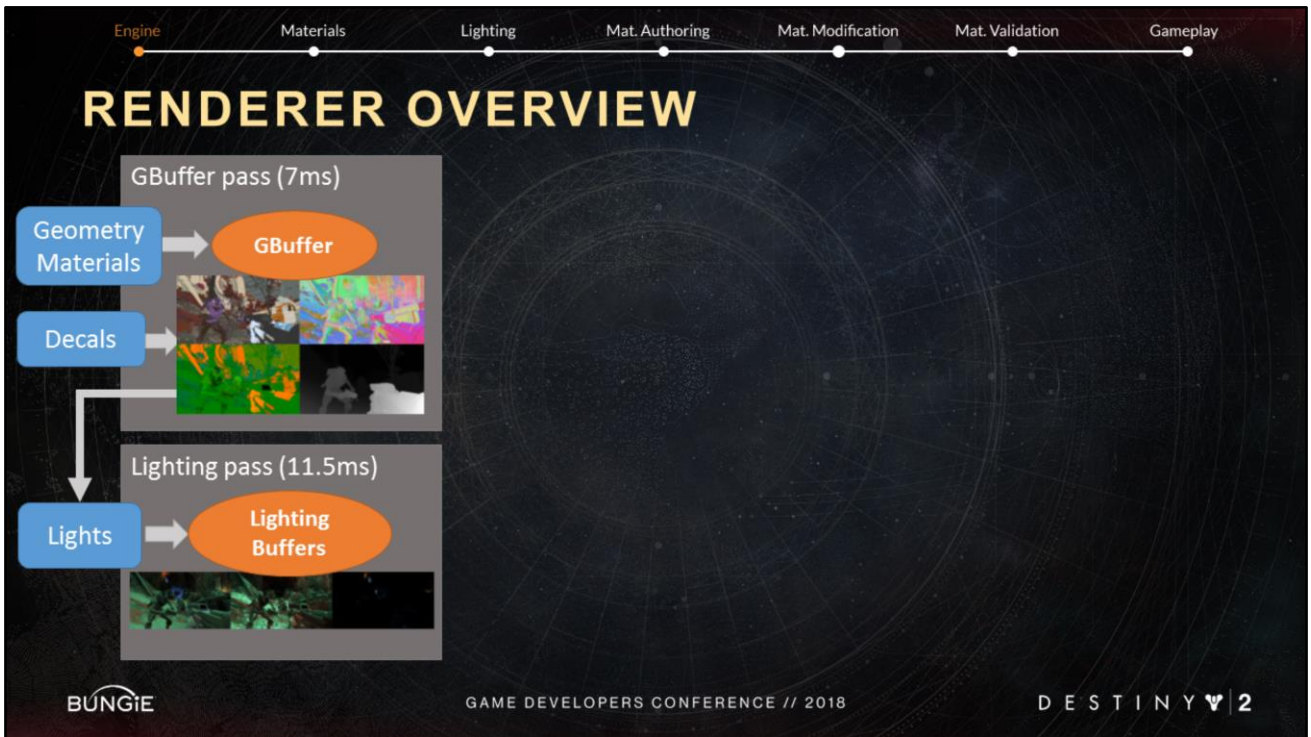
We have a large number of unique shaders and they are all complex node graphs

We support up to 50 AI

9 players in combat, 26 in social spaces, all with full customization



Rendering engine wise, let's take a look at a high level breakdown of a frame, like this one for example



Our engine is deferred

Objects/materials rendered to the gbuffer

Deferred decals rendered as projection volumes modifying the gbuffer

Light sources using part of the gbuffer applied as stenciled volumes, render to diffuse and specular lighting buffers

One thing to note is that we do not use a tiled renderer because most of our lights are unique shader nodegraphs.

Engine

Materials

Lighting

Mat. Authoring

Mat. Modification

Mat. Validation

Gameplay

RENDERER OVERVIEW

GBuffer pass (7ms)

Diffuse Lighting

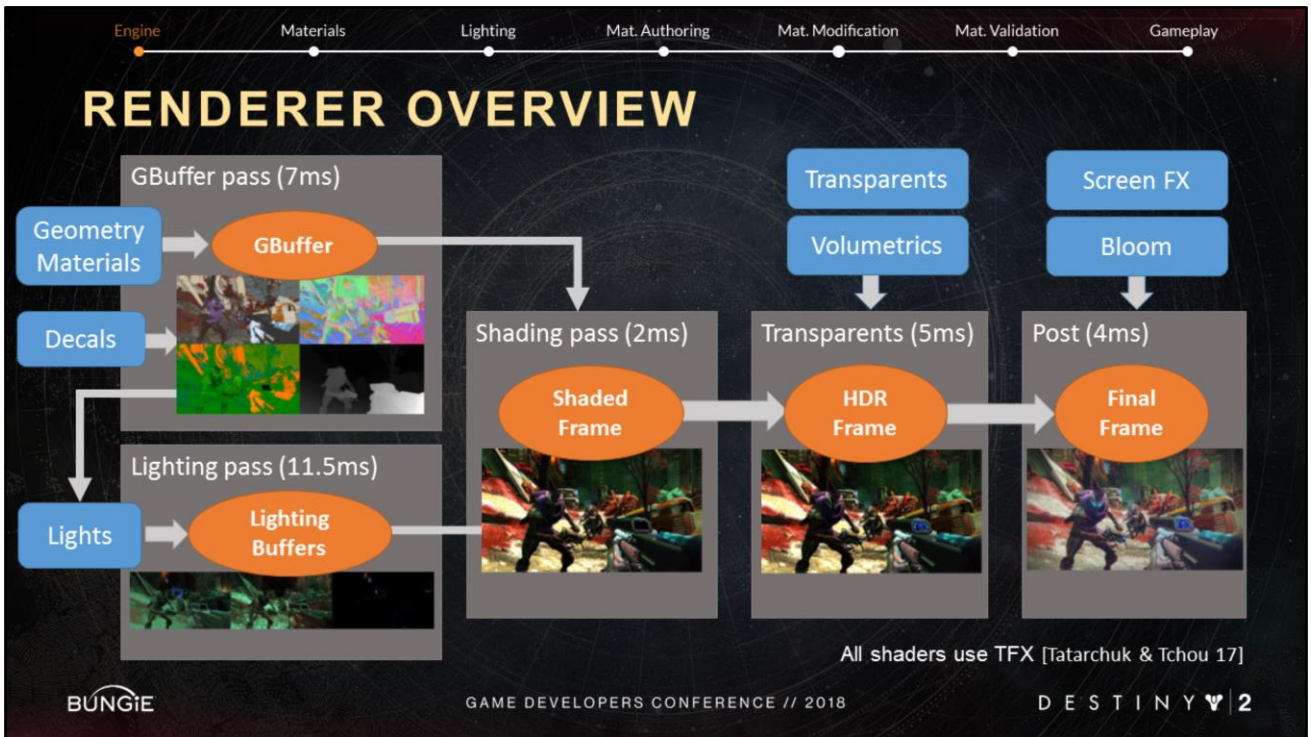
Ambient-Specular Lighting



BUNGIÉ

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2



Combine lighting information with materials in shading pass, apply atmosphere, to create the opaque shaded frame

Transparents and volumetrics forward rendered and blended on top to create the base HDR frame

Postprocess pipeline: with FX such as bloom, DOF, color correction that all create the final frame

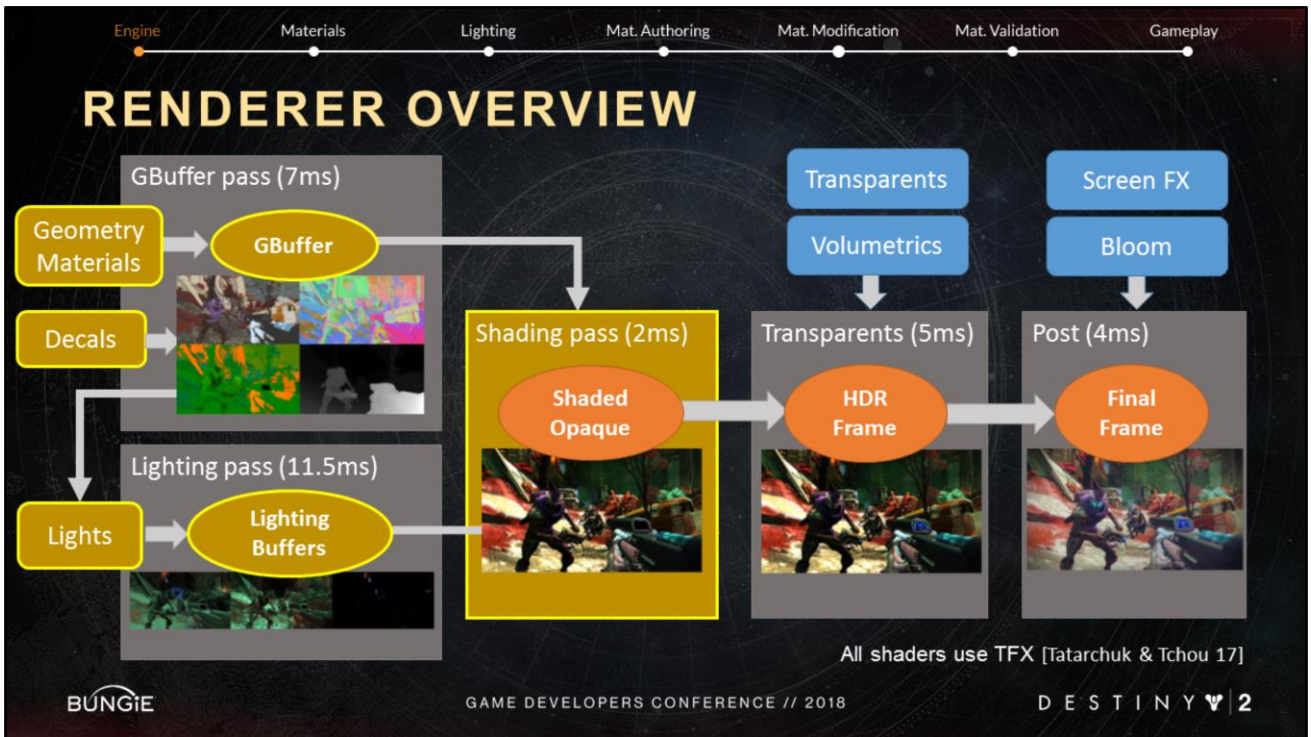
Shader system TFX (GDC17) is used for all shaders and fixed function render passes

(Note that render pass GPU budgets are for a 30FPS target)

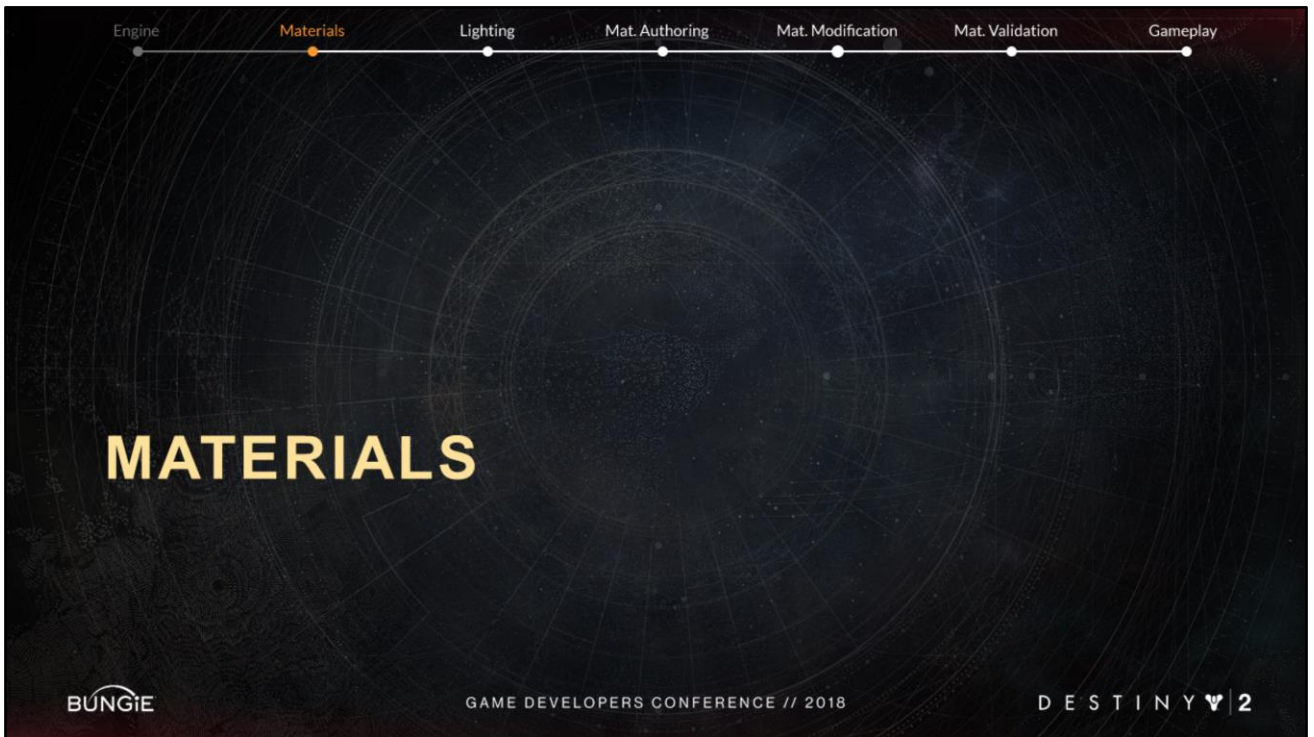




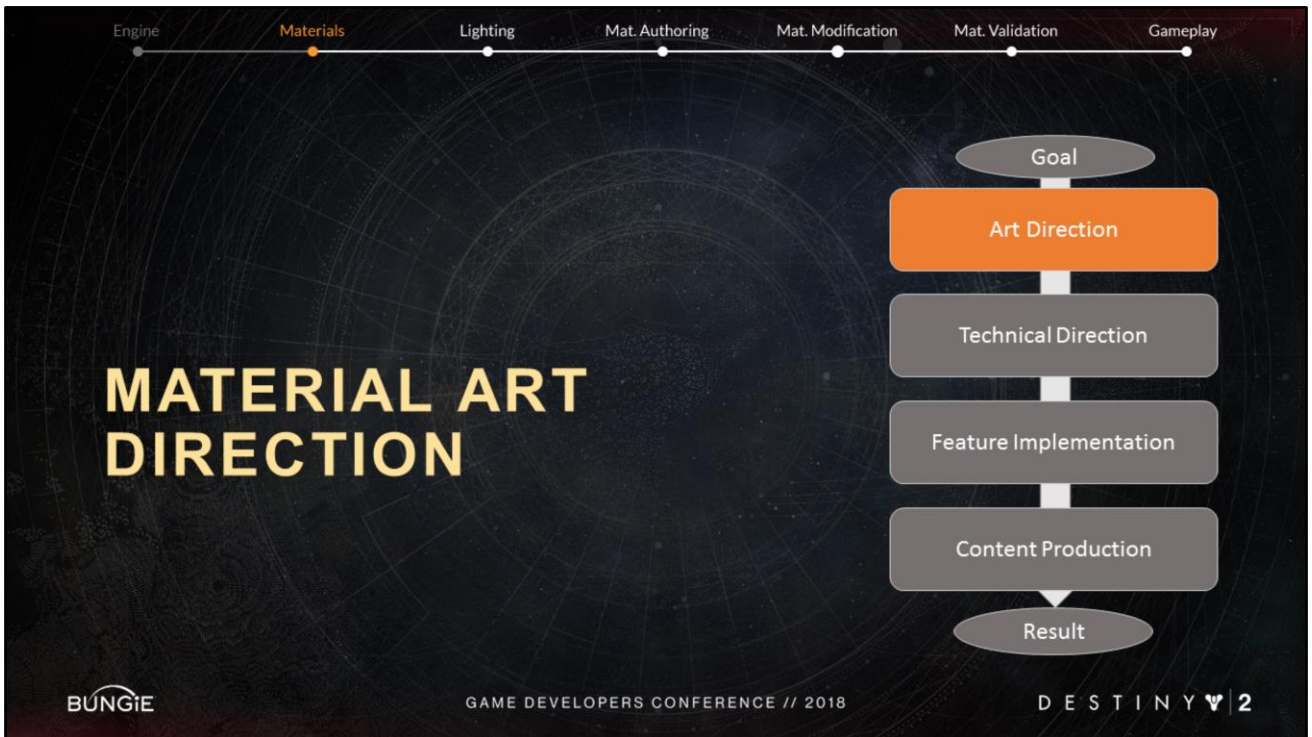




Today we will be covering those highlighted sections.



Our journey started with materials



Starting with art direction defining our visual goals

MATERIAL ART DIRECTION

• Terrestrial

- Stone
- Foliage
- Snow
- Metal
- Wood
- Plastic
- Cloth

+ Wetness



Metal



Stone



Wood



Plastic



Cloth



Foliage



Snow

For Destiny 2, we wanted to improve our material definition, specifically for terrestrial, real world materials.

We build worlds, so we need things like stone, foliage, snow.

We are a game about weapons and armor made of metal, wood or plastic.

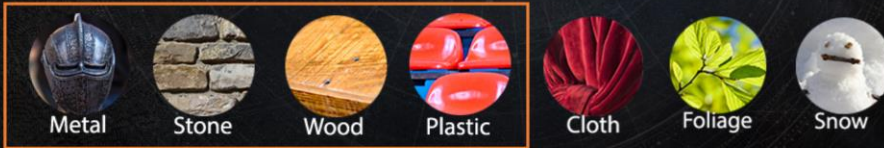
Our characters also wear lots of capes and robes made of cloth that show both fuzzy and translucent properties.

All those real world materials can get wet, which is something Nate will develop more later in this talk.

MATERIAL ART DIRECTION

- Terrestrial

- Stone
- Foliage
- Snow
- Metal
- Wood
- Plastic
- Cloth



For the purpose of this talk I will group stone metal wood and plastic as one category that we will call common materials.

Engine

Materials

Lighting

Mat. Authoring

Mat. Modification

Mat. Validation

Gameplay

MATERIAL ART DIRECTION

- Terrestrial
 - Stone
 - Foliage
 - Snow
 - Metal
 - Wood
 - Plastic
 - Cloth



Common



Cloth



Foliage



Snow

BUNGE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

MATERIAL ART DIRECTION

- Magic, alien, sci-fi
 - Iridescent
 - Light sources
 - Space Magic



Iridescence

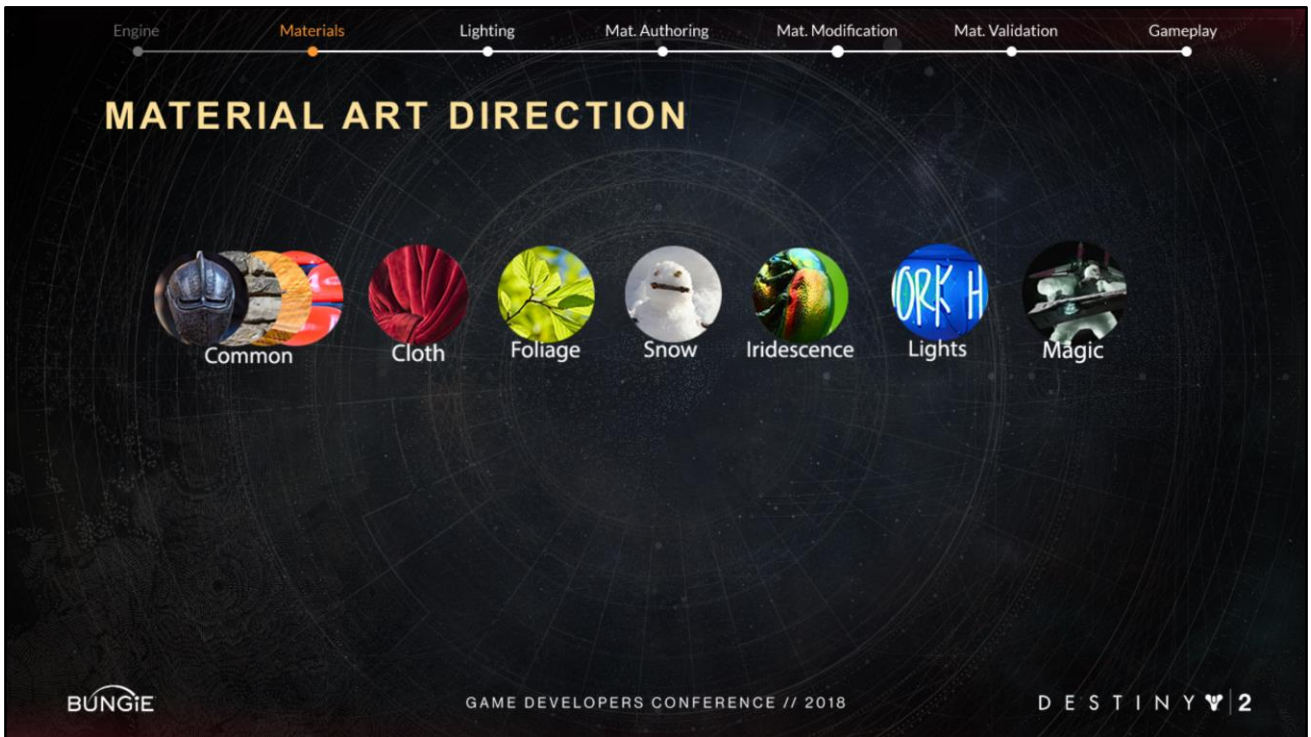


Lights

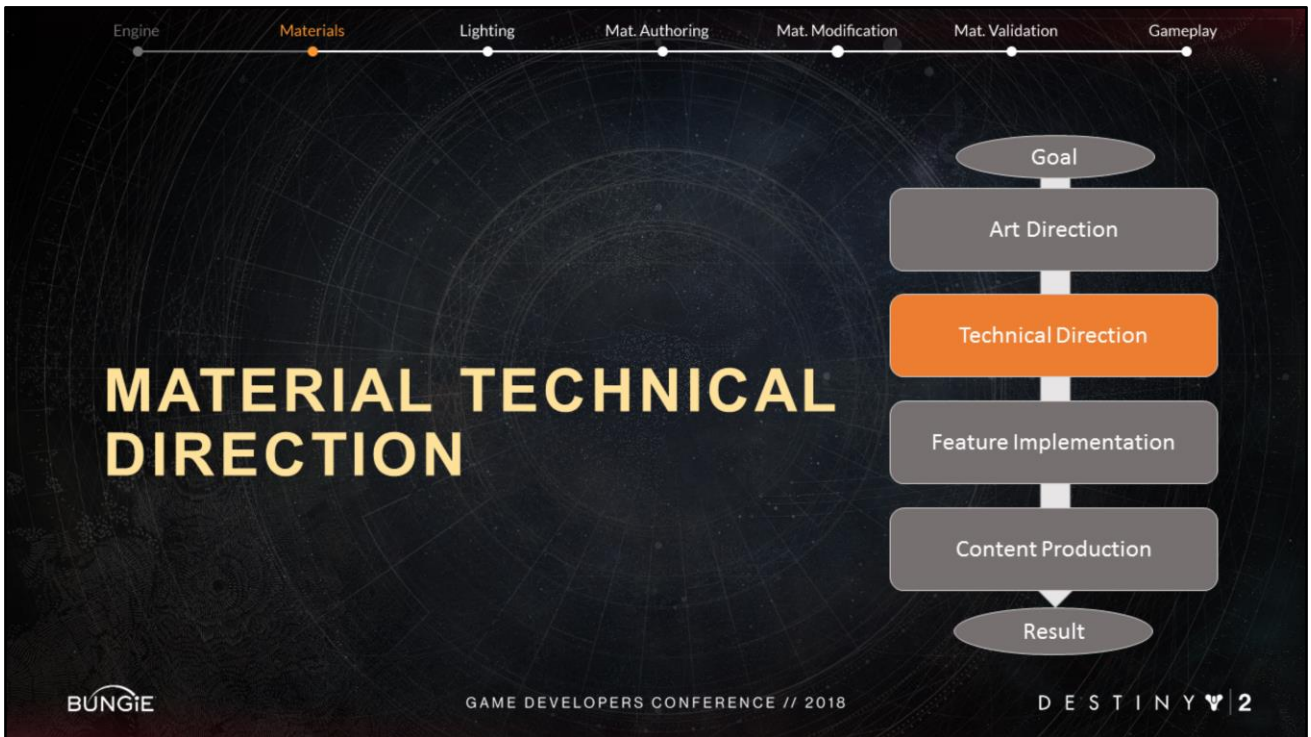


Magic

We also need the ability to express magic as well as alien and futuristic materials which are at the core of the Destiny franchise look:
Iridescent surfaces like beetles,
Light sources like LEDs and neon lights,
And just... magic in general?



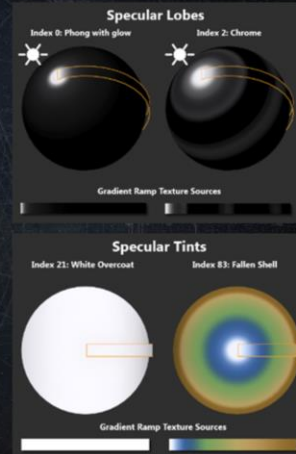
And this is our line up of materials as visual goals!



And now that our goals are clear, we need to look at technology, let's begin with our starting point
Which is the original Destiny engine.

ORIGINAL DESTINY MATERIAL MODEL

- Albedo
- Smoothness
- Texture driven shading model
 - Shape of specular highlight
 - $N \cdot V$ based specular color
- Material index
- Branch for glow
 - Repurpose smoothness as intensity



[Tatarchuk 13] [Tatarchuk & Wang 14]

In the original Destiny our shading model was mostly content driven:

Common albedo smoothness

Texture driven specular highlight shape

Specular color parameterized by $N \cdot V$ (glancing angle) defined by a color ramp

Stored as look up tables, indexed by material ID

We would branch based on specific material IDs to repurpose smoothness as glow intensity

If you want to know more about this shading model it was presented by Natalya Tatarchuk at SIGGRAPH in 2013 and 2014

Engine

Materials

Lighting

Mat. Authoring

Mat. Modification

Mat. Validation

Gameplay

ORIGINAL DESTINY MATERIAL MODEL

- Low memory footprint, low bandwidth cost
- Performant on all generations
- Can express a large variety of materials

BUNGE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

Advantages:

Compact (single material index), efficient (memory and perf)

Scalable to two generations of consoles (original Destiny also shipped on ps3 and xbox 360)

Express wide variety of materials

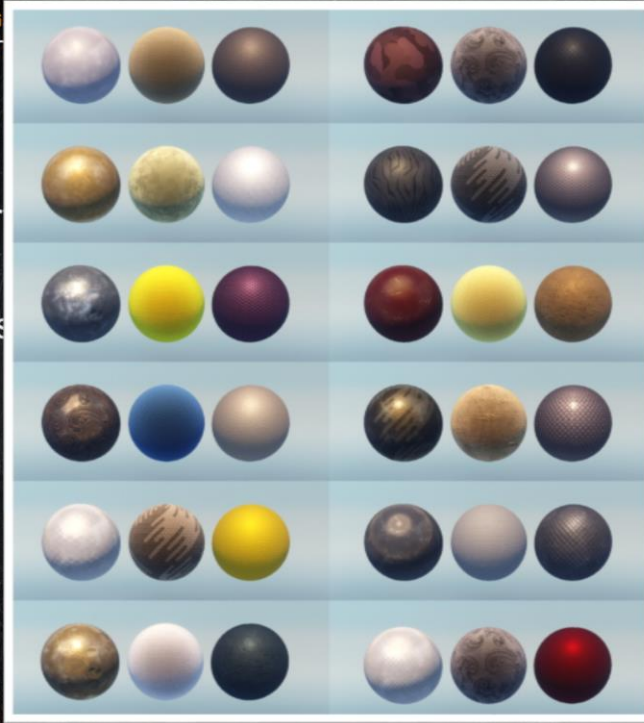
Engine

Material

ORIGINAL

- Low memory
- Performant
- Can express

BUNGE



Validation

Gameplay

DESTINY 2

Engine **Materials** Lighting Mat. Authoring Mat. Modification Mat. Validation Gameplay

ORIGINAL DESTINY MATERIAL MODEL

- Low memory footprint, low bandwidth cost
- Performant on all generations
- Can express a large variety of materials
- Easy to make alien looking surfaces

Common ? Cloth ~ Foliage Snow ~ Iridescence ✓ Lights ✓ Magic ✓

BUNGE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

Biased towards exotic and alien looking surfaces

If we look at our goals lineup:

Glow could be used for magic, lights, also used to approximate snow scattering

Texture based specular gave us iridescence and view based Fresnel effect for cloth

We didn't have any concept of translucency for foliage

What about more common materials?

Engine

Materials

Lighting

Mat. Authoring

Mat. Modification

Mat. Validation

Gameplay

ORIGINAL DESTINY MATERIAL MODEL

- Material expressivity is bottlenecked by authoring cost (mud?)
- Decision making is tedious

BUNGE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

That's where we hit our bottleneck.

Back during the original Destiny development I tried to help an artist make a mud shader. Picking the correct material ID was tedious (for mud we gave up and found a workaround with an albedo cubemap)



ORIGINAL DESIGN

- Material expression
- Decision making

You had to choose one of those sphere labelled with technical terms (diffuse tinted 10% or white overcoat 20%) that don't relate to known materials

ORIGINAL DESTINY MATERIAL MODEL

- Material expressivity is bottlenecked by authoring cost (mud?)
- Decision making is tedious
- New materials requires technical knowledge
 - Highly dimensional
 - Possible to break physical rules
 - No standards
- **Too many controls** to create a known material
- Non-blendability

If you couldn't find what you needed, creating a new one required more technical knowledge
High dimensional problem solved with content
Non-energy conservative materials
No real world standards to follow as template
Too many controls to create a known material

Because of the material index, the shading model was non-blendable, creating complicated workflows and workarounds that often didn't scale or underutilized the feature set

Some people might think: isn't more control good?
Let me show you a quick example illustrating where this strategy fails



In the game QWOP, your goal is to run forward for 100m, and for this you have to control each leg muscle independently with the keys Q,W,O,P.

Even with a simple goal, which could be achieved with a single key control, the run forward key, the task can be made near impossible by just having too many parameters exposed.



So what about style? You don't always want the same boring run.
If you wanted to make the game easy, first guarantee that you can move forward and not fall,
then add style controls.

Engine Materials Lighting Mat. Authoring Mat. Modification Mat. Validation Gameplay

PRIORITIZE

- Define what matters
- Common and important things should be easy to do or make

BUNGIE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

In a way we applied this same concept to our material parameters.
We prioritized our goals and lowered complexity required to make most common and important things.

MATERIAL TECHNICAL DIRECTION

- Intuitive material model
- Recreate real world materials
- Adding variety doesn't break simplicity
- Separate real world and space magic
- Blendable per-pixel

From there we knew we wanted an intuitive material model, capable of recreating all desired materials in order of priority.

Most used materials should be easy to make and variety should not break simplicity.

Some artists were worried to lose control if we focused on realistic materials, but the goal was to actually give them more room to express magic and make it look naturally distinct from familiar objects.

Blendable per pixel was a strong workflow desire, allows layering and combinations of all materials so natural force multiplier.

MATERIAL TECHNICAL DIRECTION

- Physically Based Rendering
- Disney BRDF [Burley 12]
- What we liked about PBR metalness workflow

Safety	Hard to break
Simplicity	Quick decision making
Guidance	Measured parameters, standards

Luckily, a large part of the industry was already switching their content pipeline to Physically Based Rendering and specifically its material workflow. We were interested in the Disney metalness gloss workflow described by Brent Burley in 2012.

This talk will not go over the basics of PBR, but I'd like to mention a few properties that we liked:

Safety	Hard to break, energy conservation
Simplicity	Expressive with only a few parameters, quick decision making
Guidance	Measured parameters, standards for artists to follow, education easier

Those pillars are at the core of how we extended our model to add material variety while keeping an intuitive workflow.

Engine Materials Lighting Mat. Authoring Mat. Modification Mat. Validation Gameplay

MATERIAL FEATURE IMPLEMENTATION



PBR FOUNDATIONS

- Metalness, Albedo, Smoothness
- Smith-GGX [Lagarde 14]
- Retro reflective diffuse (Disney BRDF)

We started with a now fairly standard Metalness, Albedo, Smoothness model that is really good at representing common materials.

We also switched from a texture driven lobe to GGX, using the Smith visibility function as described in Sebastien Lagarde's PBR talk at SIGGRAPH 2014, and added retro-reflection from the Disney model I referenced earlier.

Engine

Materials

Lighting

Mat. Authoring

Mat. Modification

Mat. Validation

Gameplay

METALNESS, ALBEDO, SMOOTHNESS



BUNGE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

Common materials (metals, plastics, concrete) are the majority of the surfaces you are seeing in this hangar.

Engine Materials Lighting Mat. Authoring Mat. Modification Mat. Validation Gameplay

METALNESS, ALBEDO, SMOOTHNESS

Metalness Albedo Smoothness

BUNGE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

The image displays a presentation slide for the 'Materials' section of a game development pipeline. At the top, a horizontal timeline lists the stages: Engine, Materials (highlighted with an orange dot), Lighting, Mat. Authoring, Mat. Modification, Mat. Validation, and Gameplay. The main title 'METALNESS, ALBEDO, SMOOTHNESS' is centered in a large, bold, yellow font. Below the title are three side-by-side screenshots of a game environment, each showing a character in a futuristic setting. The first screenshot, labeled 'Metalness', shows the character with a metallic sheen. The second, labeled 'Albedo', shows the character with a bright, high-contrast appearance. The third, labeled 'Smoothness', shows the character with a smooth, polished look. At the bottom of the slide, the Bungie logo is on the left, 'GAME DEVELOPERS CONFERENCE // 2018' is in the center, and 'DESTINY 2' is on the right.

And this is what each parameter looks like in our gbuffer.

Engine **Materials** Lighting Mat. Authoring Mat. Modification Mat. Validation Gameplay

METALNESS, ALBEDO, SMOOTHNESS

Common ✓ Cloth Foliage Snow Iridescence Lights Magic

Metalness



In game Gbuffer



Dielectric 0.0	Blend	Metal 1.0
-------------------	-------	--------------

Albedo



In game Gbuffer




RGB 0..1

Smoothness



In game Gbuffer



0.0	Linear	1.0
-----	--------	-----

BUNGE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

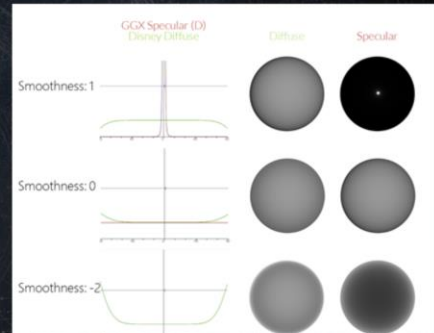
This model answers the common questions: is a surface metal or not? what color is it? how smooth is it?

The top row of spheres shows the in game render, bottom row shows our gbuffer debug visualization.

Now we can make common materials

MATERIAL: FUZZ

- Hyper rough GGX
- Allow negative smoothness: inverted GGX lobe
- Attenuate Smith visibility
- **Simplicity:** Artists control it as fuzz amount



We switched to GGX and noticed that negative smoothness inverts GGX lobe and looks fuzzy. So we added a control for making smoothness negative.

In some cases adding a separate parameter makes things simpler: artists control it as fuzz amount separate from smoothness instead of dealing with negative values

Engine

Materials

Lighting

Mat. Authoring

Mat. Modification

Mat. Validation

Gameplay

MATERIAL: FUZZ

In game



Gbuffer

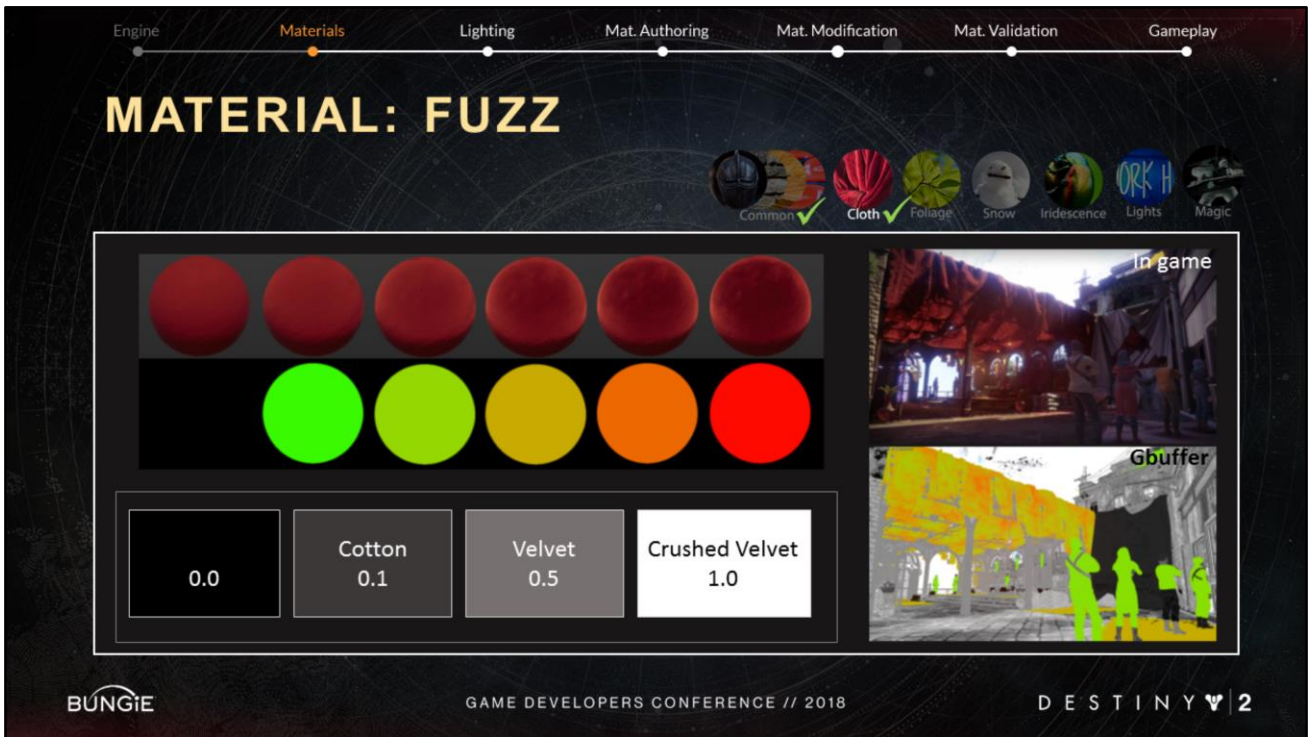


BUNGE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

Notice that the smoothness visualization is now colored to show fuzzy surfaces



Again here the row of spheres shows what the parameter looks like with a linear progression from 0 to 1

Now we can make fuzzy cloth

MATERIAL: TRANSMISSION

- Subsurface scattering and translucency
- Wrapped diffuse lobe + transmission lobe (view dependent)
- Low values brighten SSAO

```
constant_scattering =      flipped_n_dot_l * 0.4;  
forward_scattering =      pow(l_dot_v, 7.0);  
  
transmission_diffuse_lobe = saturate(constant_scattering + forward_scattering)  
transmission_diffuse_lobe *= transmission_amount;
```

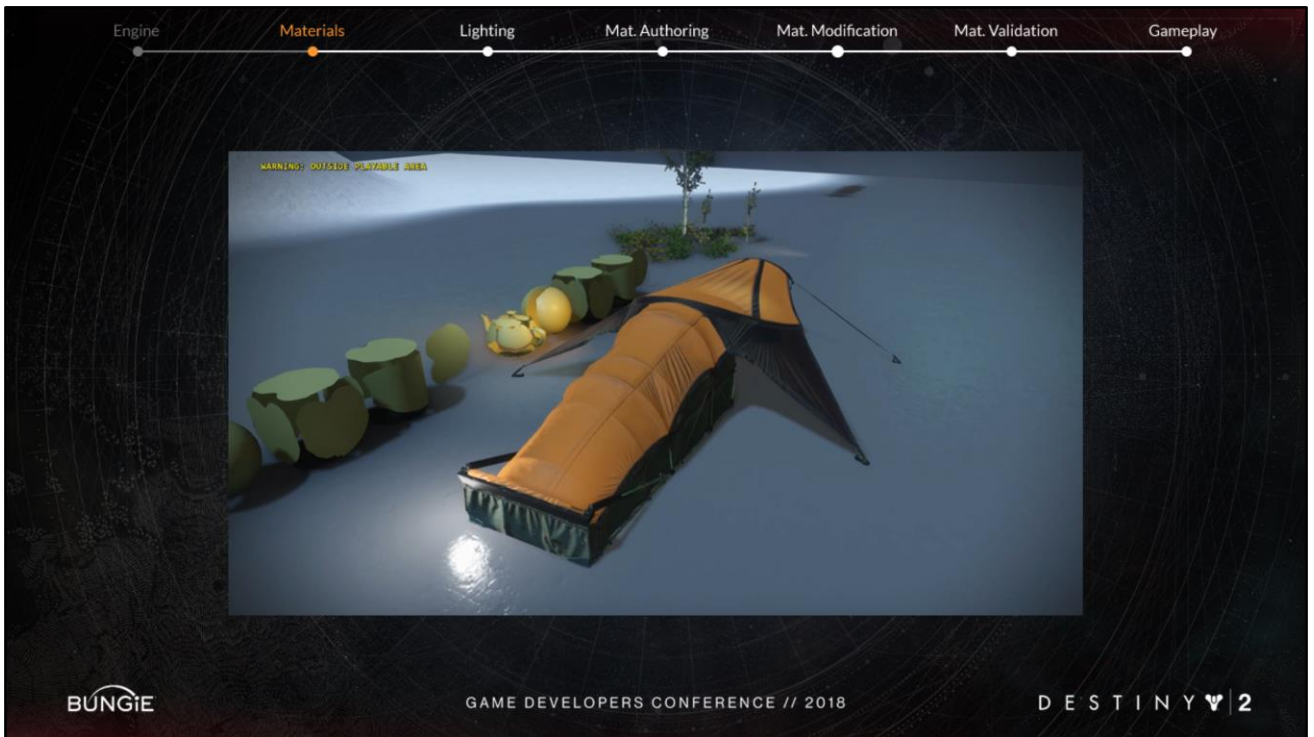
Models both sub surface scattering and translucency
Wrapped diffuse + inverted diffuse lobe with view dependence
Low range brightens SSAO to prevent it from over darkening snow creases



With transmission off, snow looks like plaster and SSAO overdarkens terrain in the distance



With transmission off, snow looks like plaster and SSAO overdarkens terrain in the distance



Thin nylon fabric

Engine

Materials

Lighting

Mat. Authoring

Mat. Modification

Mat. Validation

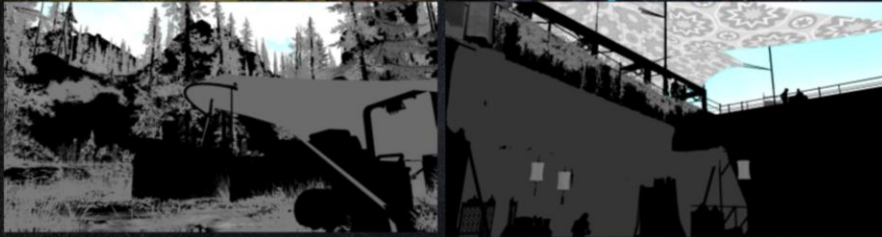
Gameplay

MATERIAL: TRANSMISSION

In game



Gbuffer



BUNGE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

We use it for tents, tarps, curtains but also foliage, fur

Engine **Materials** Lighting Mat. Authoring Mat. Modification Mat. Validation Gameplay

MATERIAL: TRANSMISSION

Common ✓ Cloth ✓ **Foliage ✓** Snow ✓ Iridescence Lights Magic

0.0 Snow 0.2 Foliage 0.3 Cloth 0.5 1.0

In game

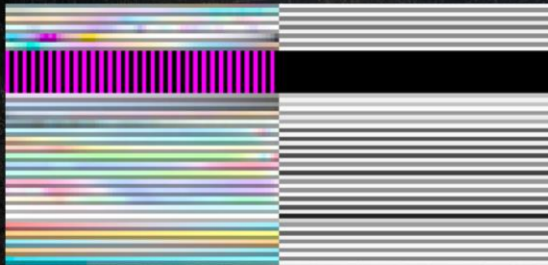
Gbuffer

BUNGE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

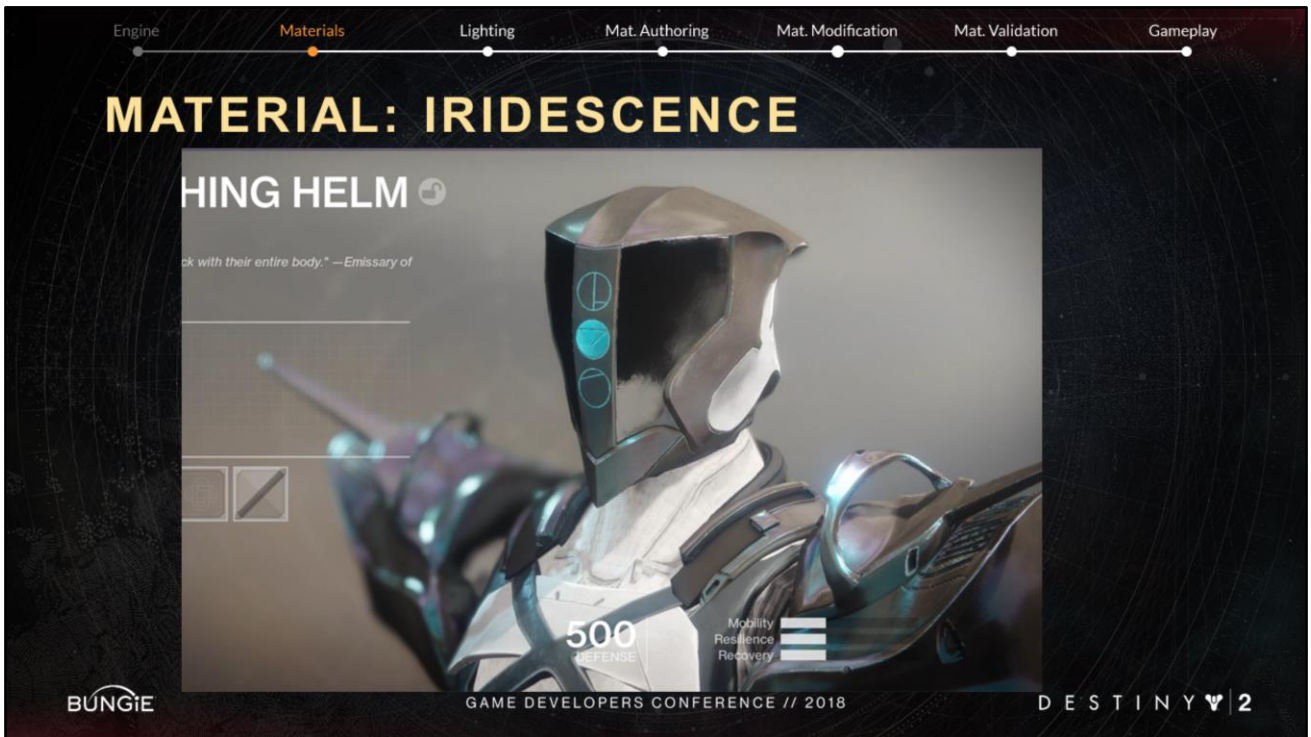
Single parameter + table of standardized values
 Now we can make foliage and snow

MATERIAL: IRIDESCENCE

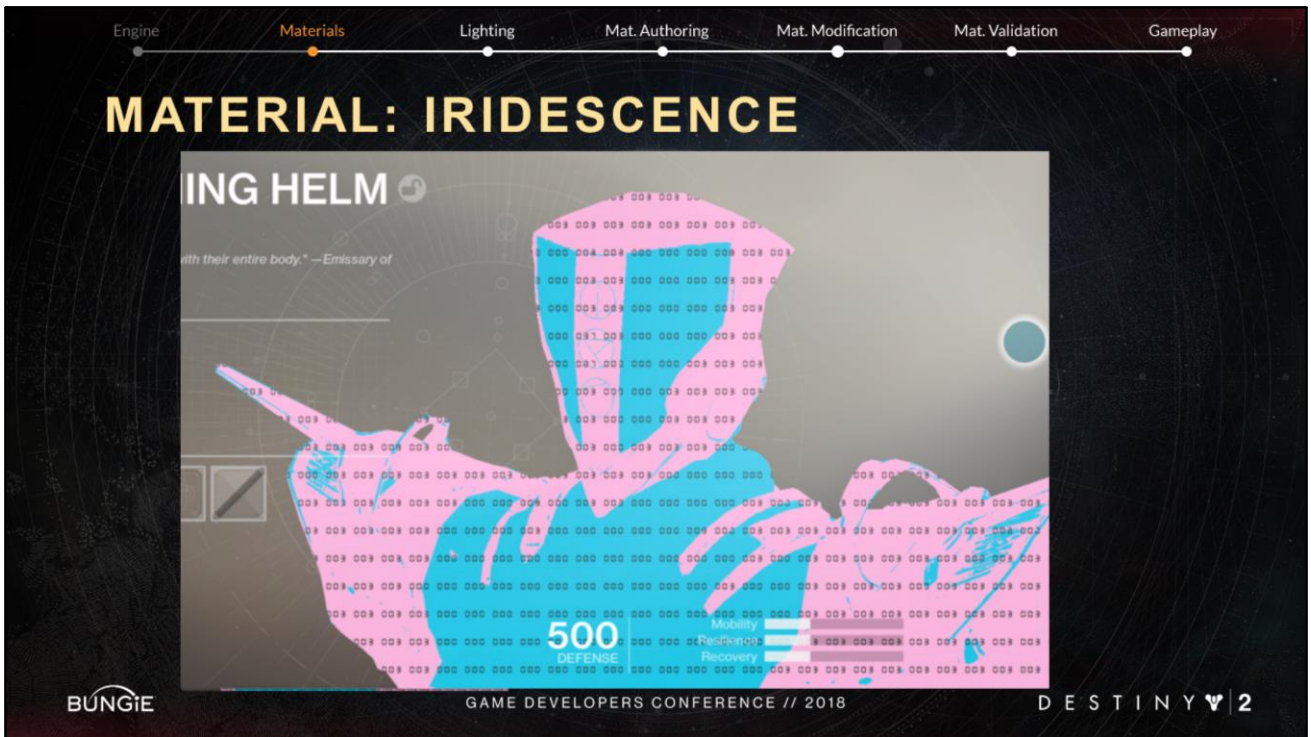
- Lookup texture (N dot V, index) that redefines specular color
- **Safe:** Attenuate diffuse by alpha for energy conservation
- **Guidance:** Model thin film interference and known iridescence types as standards



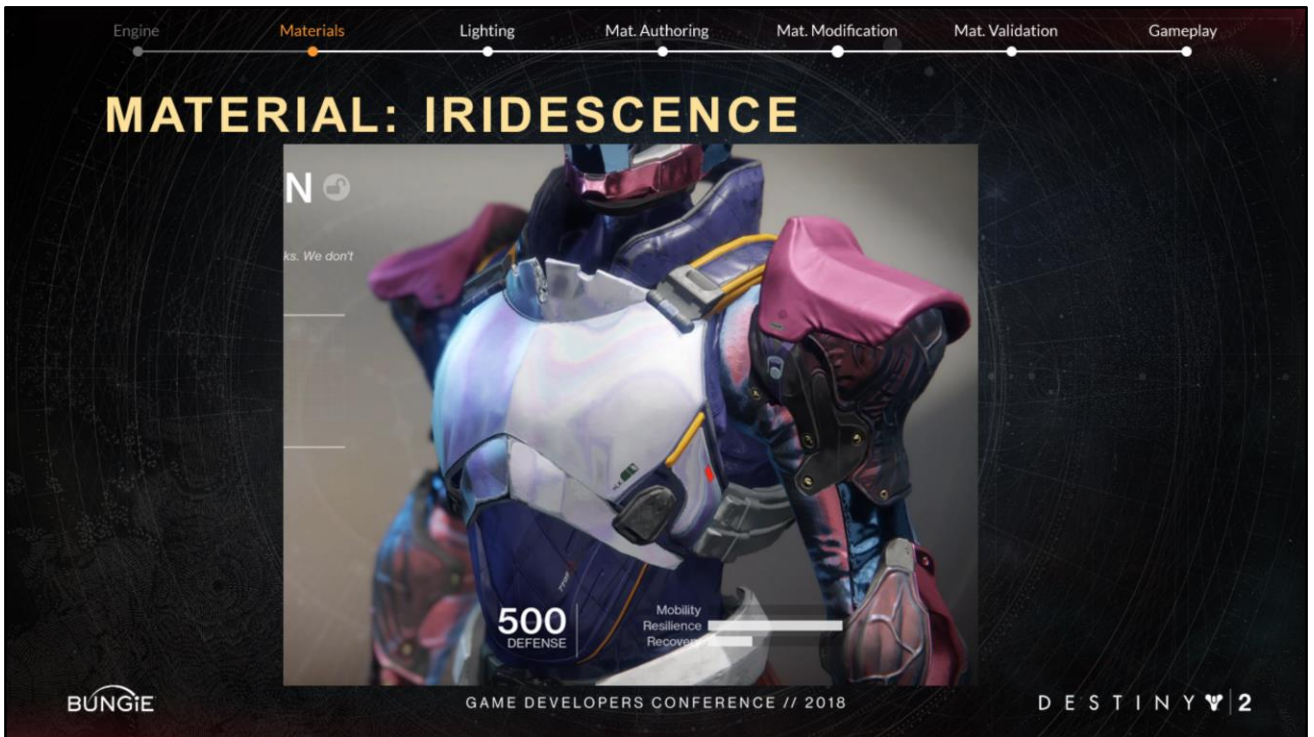
Update original Destiny texture based specular color, make it more PBR friendly
 Lookup texture (N.V, index) that redefines specular color
 Attenuate diffuse color by average specular energy for energy conservation
 Presets: thin film interference, photonic crystals



Here is an armor set showing a subtle iridescent quality



And our gbuffer debug mode that displays the iridescent material index (0 is no iridescence)



Here is another example showing 3 different types of iridescence on a single armor set

Engine

Materials

Lighting

Mat. Authoring

Mat. Modification

Mat. Validation

Gameplay

MATERIAL: IRIDESCENCE



BUNGE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

And you can see the numbers 6, 44, 45 used here

Engine Materials Lighting Mat. Authoring Mat. Modification Mat. Validation Gameplay

MATERIAL: IRIDESCENCE

BUNGIE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

We have a lot more presets that let us achieve all kinds of exotic / space magic looks

Engine **Materials** Lighting Mat. Authoring Mat. Modification Mat. Validation Gameplay

MATERIAL: IRIDESCENCE

Common ✓ Cloth ✓ Foliage ✓ Snow ✓ **Iridescence** ✓ Lights ✓ Magic ✓

Thin film 0-11 Photonic Crystals 32-57 Gradients 96-105

Even index: metal Odd index: dielectric

In game

Gbuffer

BUNGIÉ GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

Lineup doesn't show the full range because we have more
 Every even index is a metal, odd is dielectric, we need to support both since we override the specular color
 Only non-blendable parameter, restricted mostly to armor and weapons which don't need as much blending support
 Now we can make iridescent materials

MATERIAL: EMISSIVE

- Emissive = albedo * emissive intensity (additive)
- Exposed to artists
 - As monochrome intensity 0-64
 - **Simplified** workflow: HDR RGB color blended with albedo

Emissive intensity is a scaler multiplied with albedo and added on top of lighting

Exposed to artists As monochrome intensity 0-64

Simplified workflow: HDR RGB color blended with albedo

Engine

Materials

Lighting

Mat. Authoring

Mat. Modification

Mat. Validation

Gameplay

MATERIAL: EMISSIVE

In game



Gbuffer



BUNGE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

We use it for glowy plants and mushrooms, light fixtures, or illuminated billboards

Engine Materials Lighting Mat. Authoring Mat. Modification Mat. Validation Gameplay

MATERIAL: EMISSIVE

Common ✓ Cloth ✓ Foliage ✓ Snow ✓ Iridescence ✓ Lights ✓ Magic ✓

The slide features a central control panel with a color gradient bar (green to yellow) and a grayscale bar (black to white). Below these is a slider with numerical values 0.0 and 64.0. To the right, two in-game renderings are shown: 'In game' and 'Gbuffer'.

BUNGE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

This is the range we get by changing the emissive intensity from 0 to 64
Now we can make lights and magic

DESTINY 2 MATERIAL MODEL

MASTIFE

- Metalness
- Albedo
- Smoothness
- Transmission
- Iridescence
- FUZZ
- Emissive



We came up with a material model inspired by Physically Based Rendering and extended it with parameters, that even if not based on physics, give plausible results for a wide variety of real world materials. This model also allows us to express space magic, at the proper level of abstraction.

I like to call it MASTIFE.

GBUFFER: ORIGINAL DESTINY

	R	G	B	A
RT0 (8:8:8:8)	Albedo R	Albedo G	Albedo B	Vertex AO
RT1 (8:8:8:8)	Normal + Smoothness			Material ID
Depth/Stencil	Depth			Stencil

- Single material index used for lobe/tint tables lookup
- Branching based on index for emissive materials
- Smoothness encoded in the normal length

```
normal_length = 0.25 * smoothness + 0.75
```

Single material index used for lobe/tint tables lookup
Branching based on index for emissive materials
Smoothness encoded in the normal length

GBUFFER: DESTINY 2

	R	G	B	A
RT0 (8.8.8.8)	Albedo R	Albedo G	Albedo B	Iridescence ID
RT1 (10.10.10.2)	Normal + Smoothness			Blendable bit
RT2 (8.8.8.8)	Metalness	Texture AO + Emissive	Transmission	Vertex AO
Depth/Stencil	Depth			Stencil

- Increased our normal/smoothness precision
- Emissive encoded as an exponential curve $[0-1] \rightarrow [0-64]$
 - Packed in the range $[0.5-1]$ with texture AO (range $[0-0.5]$ remapped linearly to $[0-1]$)
- All non-bendable parameters in alpha channels

Iridescence stored as an integer 0-255

Emissive encoded as an exponential curve $[0-1] \rightarrow [0-64]$

Packed in the range $[0.5-1]$ with texture AO (range $[0-0.5]$ remapped linearly to $[0-1]$)

All non-blendable parameters in alpha channels because hardware can't alpha blend alpha channels.

GBUFFER OPTIMIZATION

- Constraints
 - Range and precision
 - Space (linear, gamma)
 - Lifetime and consumption
 - Blendability
 - Memory footprint
 - Performance

Fitting those parameters into a limited set of render targets is an optimization problem

Range and precision: can create banding

Space (linear, gamma, exponential)

Lifetime and consumption: which gbuffer is sampled by what and when?

Allocation/deallocation

Blendability: alpha channels are not blendable

Memory footprint: some platforms have hard memory requirements

Performance: encoding/decoding costs, bandwidth

GBUFFER OPTIMIZATION

- Tricks that helped us
 - Compression / encoding
 - Packing parameters together
 - Mutual exclusivity / repurposing
 - Index / raw bits for texture lookup or shader branching
- Worth re-optimizing the entire gbuffer



Compression / encoding

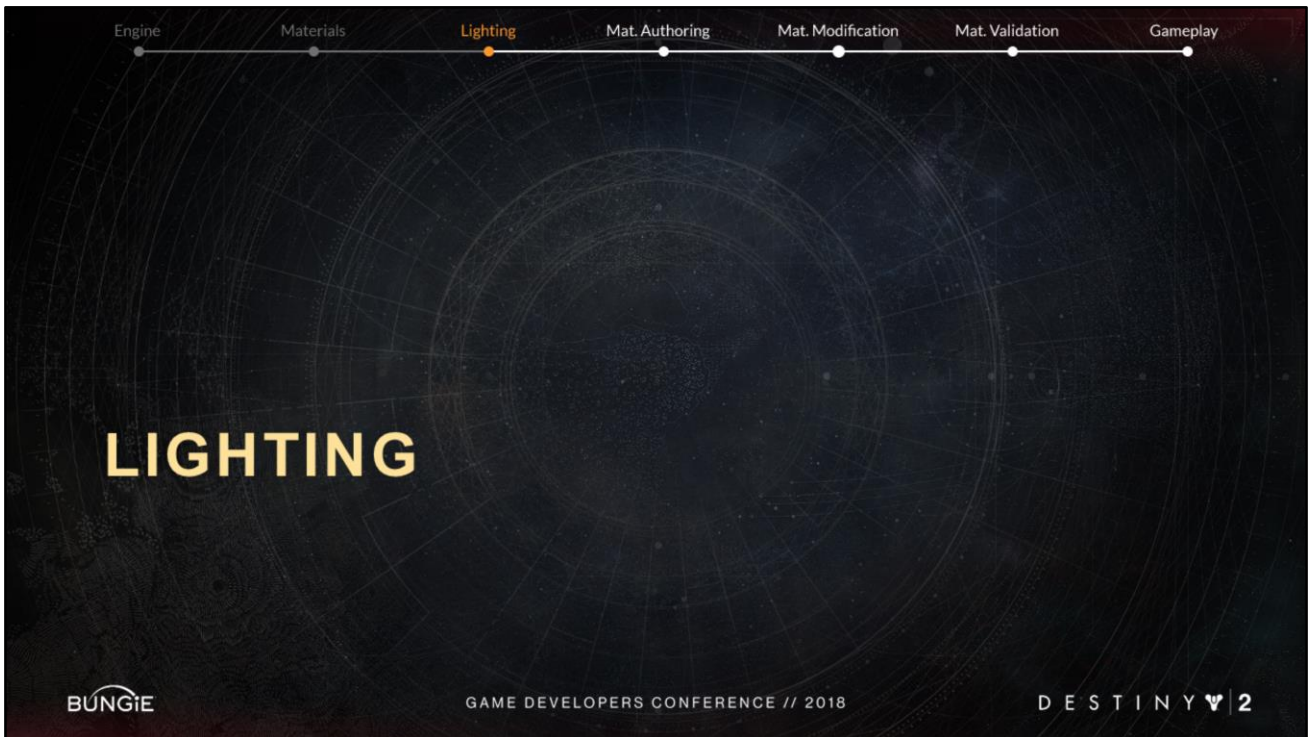
Packing

Mutual exclusivity / repurposing

Index / raw bits for texture lookup or branching

We went through a lot of different experiments

Worth re-optimizing the entire gbuffer even when adding a single new parameter



Now that we have materials, we need to light them.

Engine Materials **Lighting** Mat. Authoring Mat. Modification Mat. Validation Gameplay

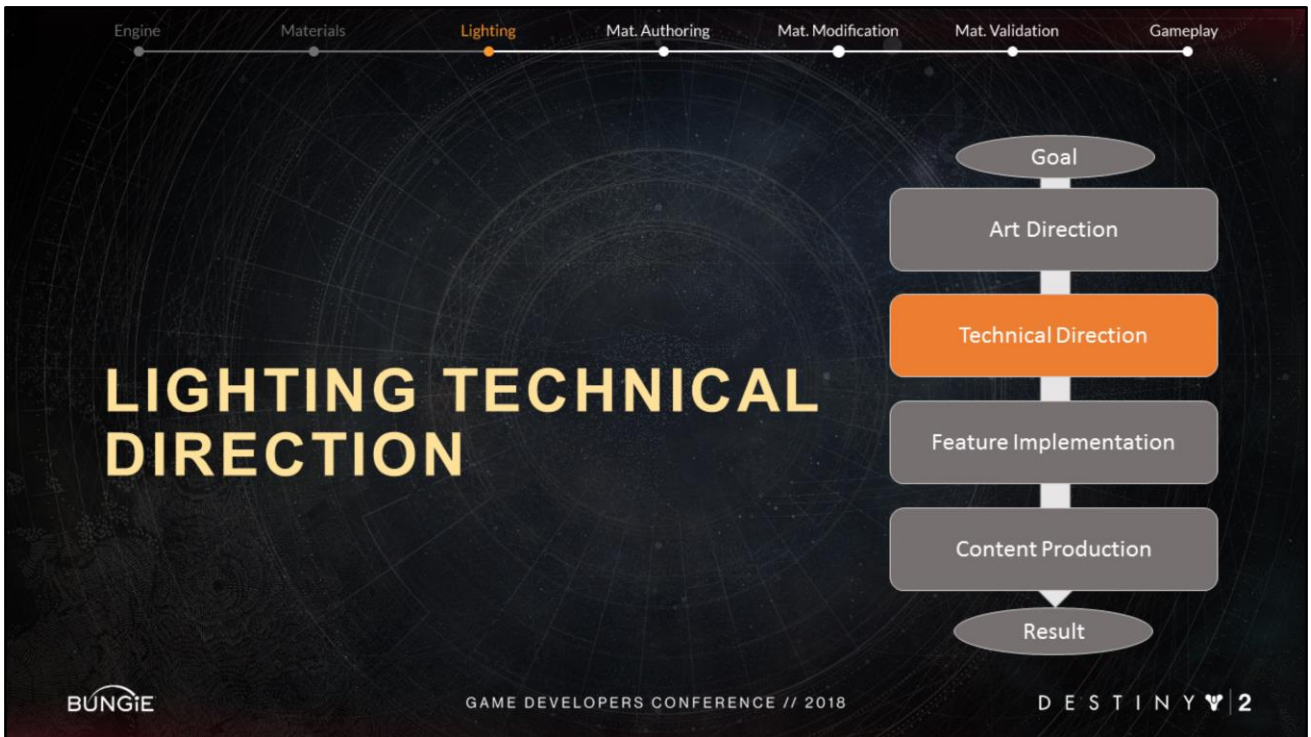
LIGHTING ART DIRECTION

- Support metals and wetness
- Extremely smooth surfaces

```
graph TD; Goal([Goal]) --> AD[Art Direction]; AD --> TD[Technical Direction]; TD --> FI[Feature Implementation]; FI --> CP[Content Production]; CP --> Result([Result]);
```

BUNGE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

Our visual goals are connected to our material goals
Support our material model, specifically metals and wet materials
Extremely smooth surfaces for a more pronounced science fiction look



We got our goals, let's talk about tech, starting with the original Destiny engine!

Engine Materials **Lighting** Mat. Authoring Mat. Modification Mat. Validation Gameplay

ORIGINAL DESTINY ENGINE

- Deferred lights
 - Point (omni), spot, line, directional (sun)
- LDR cubemap added to albedo
- N dot V Fresnel
- Ambient lighting as global up/down color

BUNGIE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

Deferred lights: point, line, directional

Direct lighting is not enough to fully express metals and wet surfaces

Specular quality lost in shadows and wherever local lights are out of range

Cubemap baked in the shader albedo (LDR)

Fresnel as N.V glancing tint

Only source of ambient lighting was a global up/down two color lobe applied to the entire world

ORIGINAL DESTINY ENGINE

- Deferred lights
 - Point (omni), spot, line, directional (sun)
- LDR cubemap added to albedo
- N dot V Fresnel
- Ambient lighting as global up/down color

Texture based highlight too rough

Low contrast, not contextual
Makes silhouettes glow

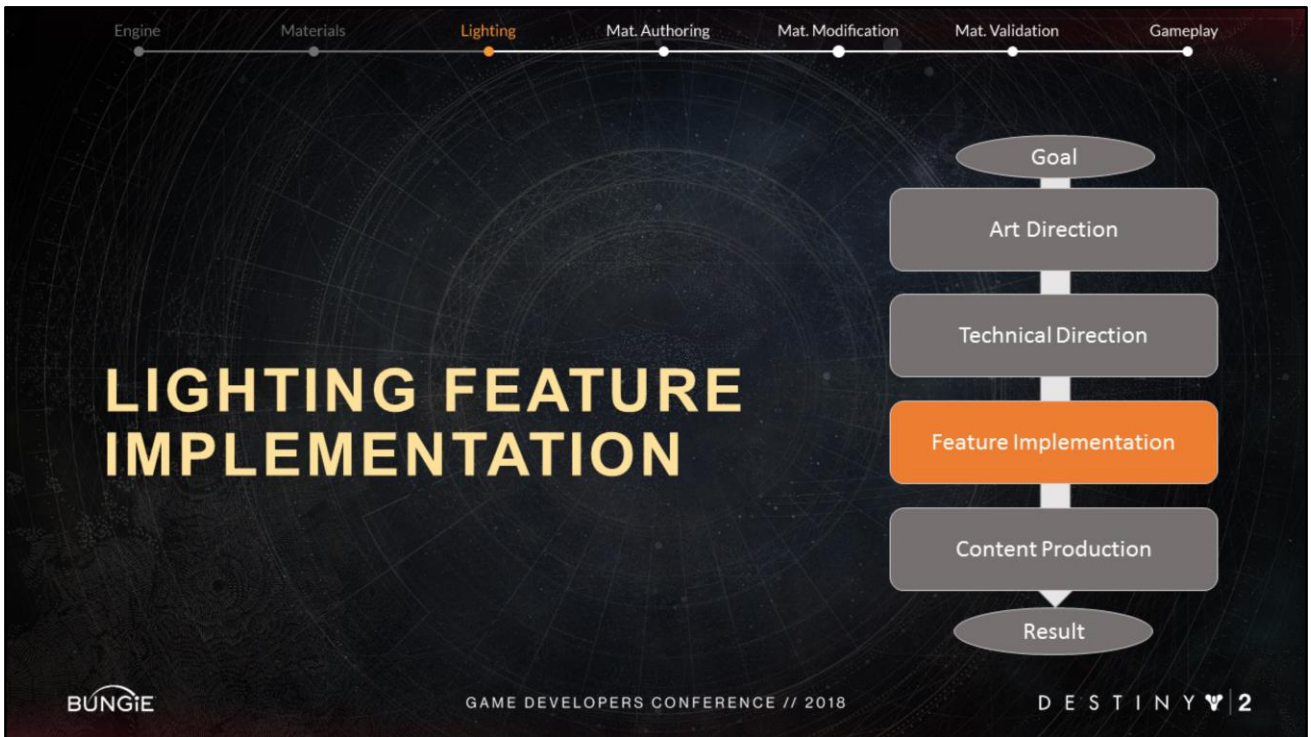
Can't represent indoors and outdoors at the same time

Texture based highlight wasn't sharp enough for high gloss.
Albedo cubemap doesn't reflect surroundings and is low contrast.
N.V Fresnel makes silhouettes of objects glow in the dark

Global ambient represents one lighting situation at a time for the entire world

LIGHTING TECHNICAL DIRECTION

- High frequency, high contrast specular
- Ambient specular solution
 - HDR Image Based Lighting

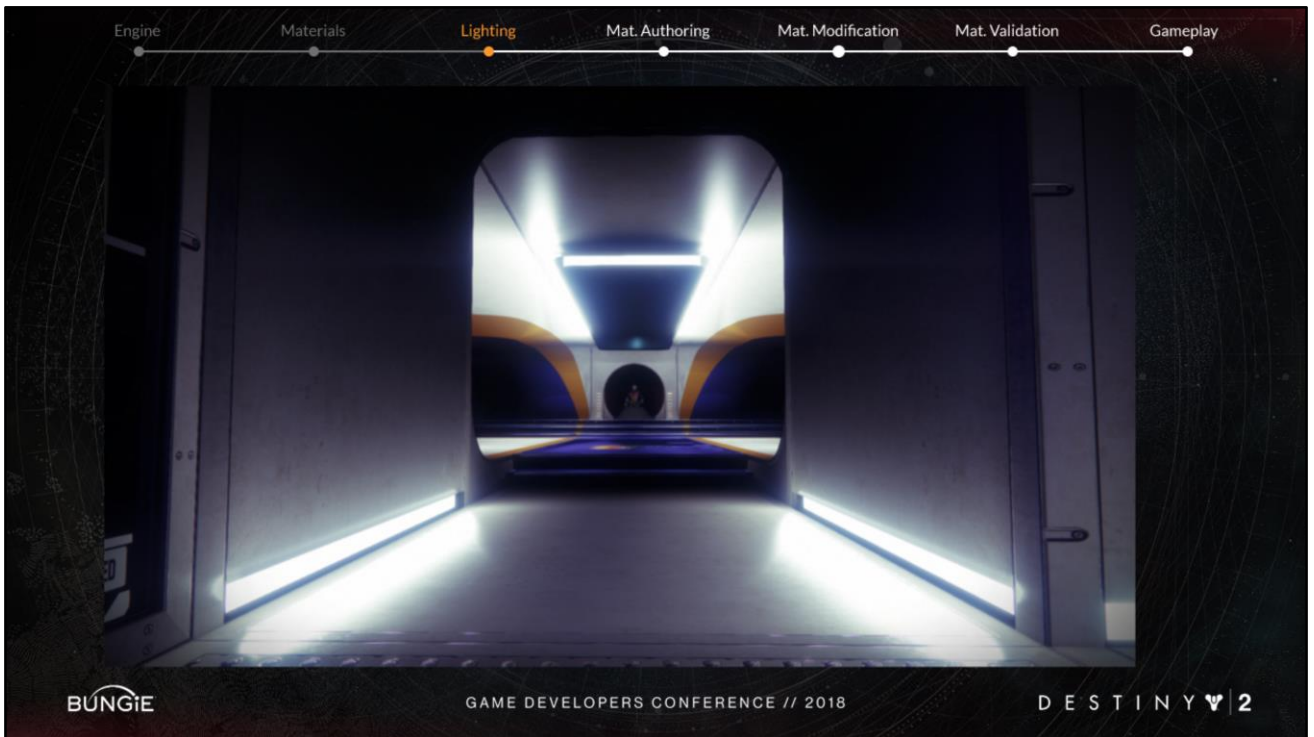


Let's talk about the lighting features we added for Destiny 2

DEFERRED LIGHTS

- Smith GGX BRDF
- Area specular lights: sphere and tube
- Light direction Fresnel

Smith-GGX BRDF: tighter specular highlight



Area lights: broader highlight without sacrificing sharpness

Engine

Materials

Lighting

Mat. Authoring

Mat. Modification

Mat. Validation

Gameplay

DEFERRED LIGHTS: FRESNEL

Light Direction Fresnel OFF



BUNGE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

Replaced N.V fresnel with Light direction Fresnel

Engine

Materials

Lighting

Mat. Authoring

Mat. Modification

Mat. Validation

Gameplay

DEFERRED LIGHTS: FRESNEL



BUNGE

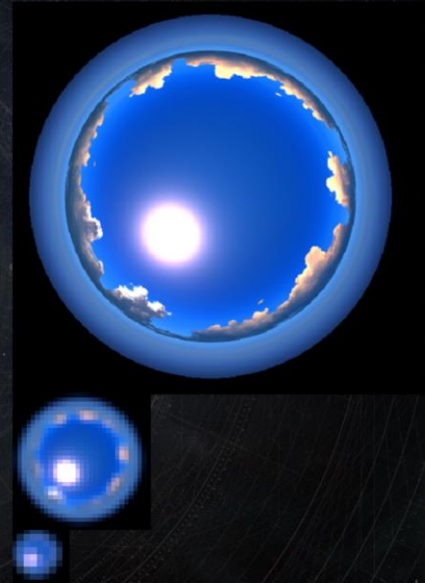
GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

Replaced N.V fresnel with Light direction Fresnel

SKY HEMISPHERE

- Render sky and clouds into hemispherical projection
- Dynamic (every 2 frames)
- Blur and calculate mip chain (gaussian)
- Apply as deferred full screen pass



We now have higher quality analytic specular, now we need IBL

Let's start with the most global light source

Render sky and clouds into hemispherical projection

Dynamic (amortized over 2 frames)

Blur and calculate mip chain to simulate smoothness (gaussian blur, cheaper than importance sampling)

Apply as deferred full screen pass to specular IBL buffer

Engine

Materials

Lighting

Mat. Authoring

Mat. Modification

Mat. Validation

Gameplay

SKY HEMISPHERE



BUNGE

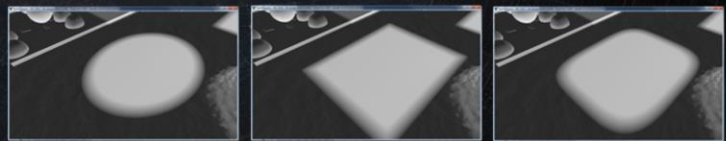
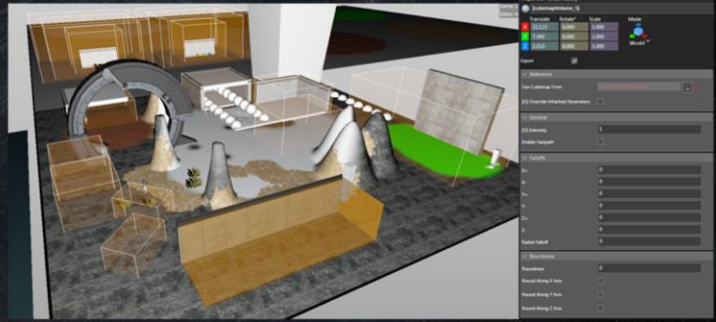
GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

Early prototype video, clouds lit and moving in the helmet reflection

LOCAL CUBEMAPS: AUTHORIZING

- Authoring in world editor
- Scaled cubes
- Fade distance
- Roundness



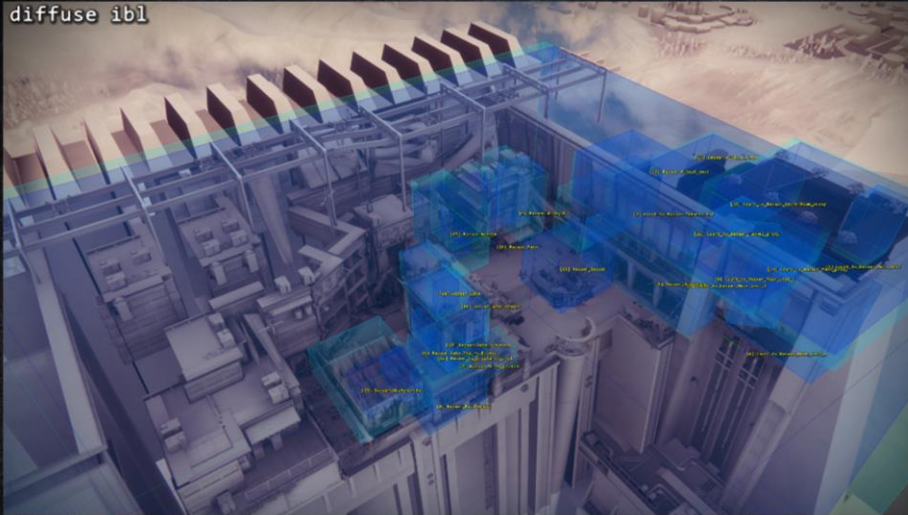
Let's go more local!

Scaled/rotated cube volumes hand placed by artists in editor

Fade distance per cube face to feather lighting contribution

Roundness parameter to match less regular, more curved environment

LOCAL CUBEMAPS: AUTHORIZING



Distribution of cubemap volumes in the tower social space

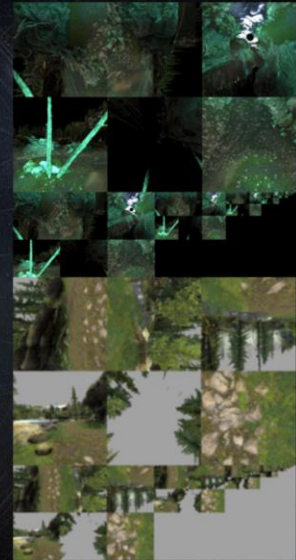
LOCAL CUBEMAPS: AUTHORIZING



More natural environment, European Dead Zone, cubes are less aligned and rotated more arbitrarily

LOCAL CUBEMAPS: BAKING

- Capture in engine (offline)
 - Override game settings
 - Neutral ambient lighting
- Filtered on GPU with importance sampling
- Encode GGX smoothness in MIP chain
- Encode sky mask as alpha

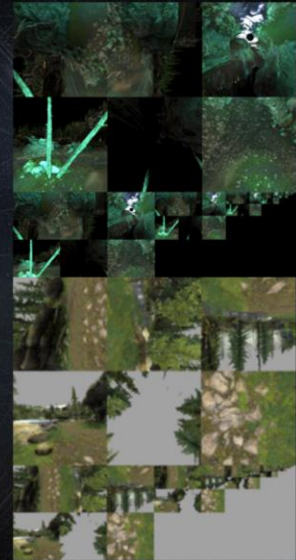


- Capture offline in engine
- Override game settings
- Disable features (particles)
- Neutral ambient lighting (for relighting later)
- Apply GGX filter on the GPU
- Encode smoothness in mip chain
- Encode sky mask in alpha channel when outside

LOCAL CUBEMAPS: BAKING

- Capture in engine (offline)
 - Override game settings
 - Neutral ambient lighting
- Filtered on GPU with importance sampling
- Encode GGX smoothness in MIP chain
- Encode sky mask as alpha

Interior (HDR RGB)	BC6H
Transition (HDR RGB + HDR Sky with alpha)	BC6H+BC4
Exterior (LDR RGB + HDR Sky with alpha)	BC7



Texture compression modes depending on where the cubemap is captured
Most used are BC6H and BC7

LOCAL CUBEMAP: RENDERING

- Applied as stenciled deferred volume
- Tint with sun and average sky color
- Blends on top of sky lighting with alpha mask
- Smaller volumes override bigger ones

Applied as stenciled deferred volume, same as lights
Tint with sun and average sky color to approximate time of day
Blends on top of sky lighting with alpha mask

LOCAL CUBEMAP: RENDERING



Let's look at an example, our character is standing on a crate inside of this half broken building. An artist placed a cubemap volume there



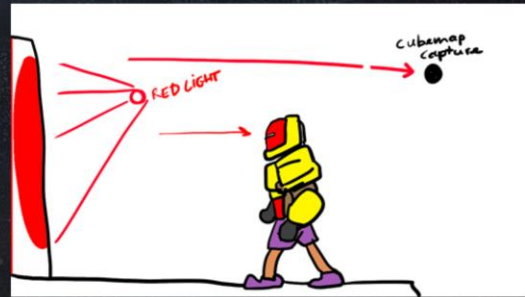
We apply sky lighting first



Then blend our local cubemap on top, which only leaves sky reflections where it is visible in the cubemap.

LOCAL CUBEMAPS: DIFFUSE LIGHTING

- Local ambient specular but global ambient diffuse



So we were happy with the results but started seeing things like this.

In this example this character has a silver visor that looks red while his painted armor doesn't. This drawing shows you what the scene looked like from a different angle.

In front of the character there is a wall that looks bright red because of a spot light but the character is out of range of the light. This wall gets captured in the cubemap, then reflected by the visor but not visible on low specular surfaces like the yellow armor.

We realized we decoupled specular from diffuse lighting because we localized ambient specular through cubemaps but our ambient diffuse was still global.

LOCAL CUBEMAPS: DIFFUSE LIGHTING

- Render last MIP level as diffuse lighting
- Render sky last MIP to diffuse too

Render last mip level of local cubemaps and sky hemisphere as diffuse lighting



Sky Diffuse Lighting





LOCAL CUBEMAPS: DIFFUSE LIGHTING

- Specular and diffuse are coupled again!



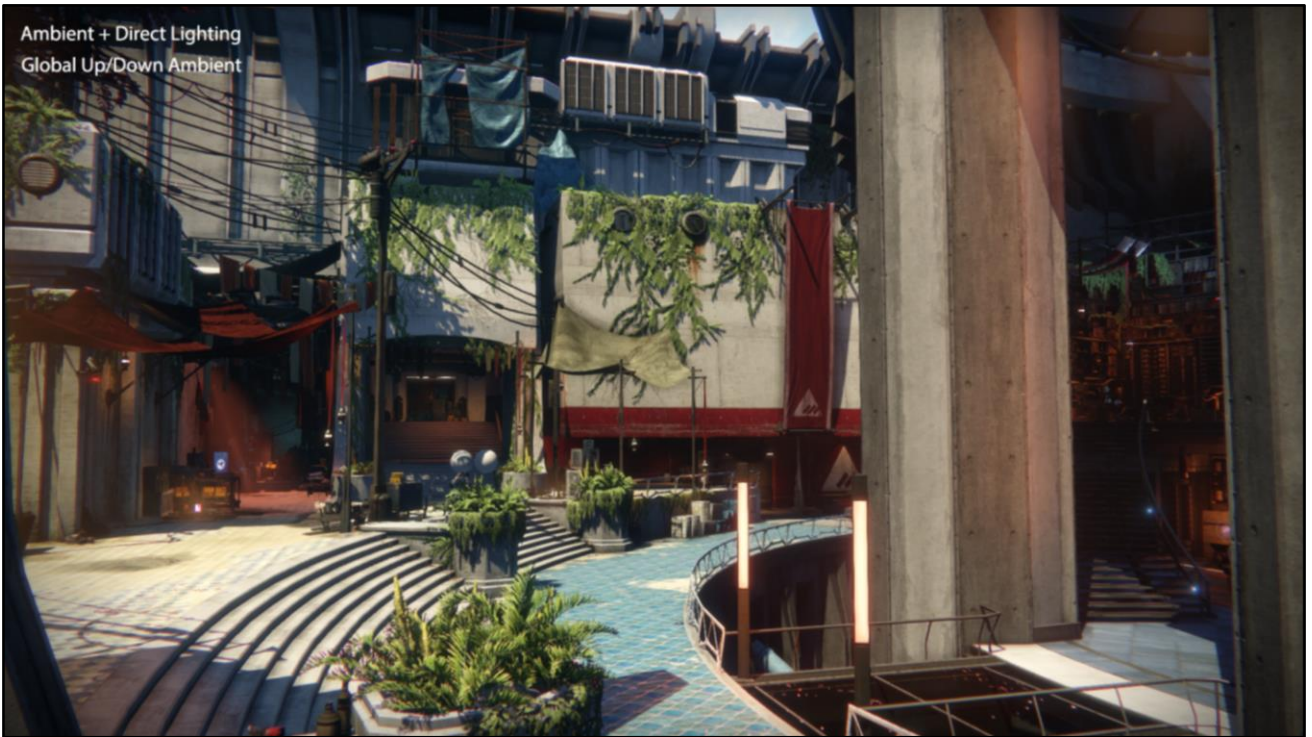
And now that guy is happier 😊



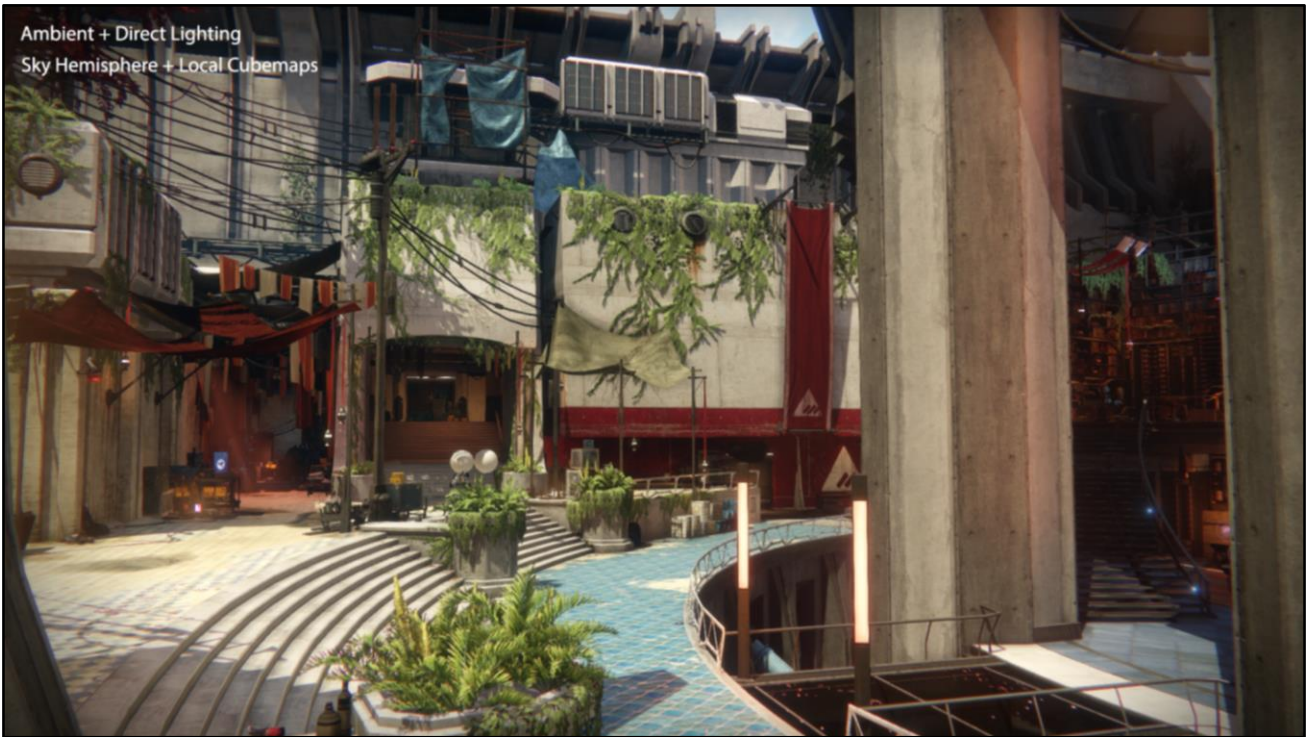
Comparison from an early prototype showing the difference between our global ambient from the original Destiny engine, and our new features: sky blended with local cubemaps



Comparison from an early prototype showing the difference between our global ambient from the original Destiny engine, and our new features: sky blended with local cubemaps



When we add direct lighting on top, notice shadows go from blue everywhere to a more natural bounce color



When we add direct lighting on top, notice shadows go from blue everywhere to a more natural bounce color

LOCAL CUBEMAPS: DIFFUSE LIGHTING



Now that we are applying cubemap volumes to diffuse, we have a new problem. Boundaries between interiors and exteriors are a lot more obvious in diffuse than specular because it's not view dependent. We went back to drawing board and added another feature.

DIFFUSE PROBES: AUTHORIZING / BAKING

- Subdivide cubemap volumes into voxels
- Capture full cubemap at each voxel, only keep last MIP
- Store local diffuse lighting as tint in volume texture
- Sky visibility as alpha channel
- Usually 6x6x3

Diffuse probes:

Subdivide cubemap volumes into voxels

Capture cubemap at every voxel location

Store average of last MIP from all directions into volume texture

Use the alpha channel sky mask as diffuse sky visibility

Average voxel resolution 6x6x3

DIFFUSE PROBES: RENDER



With diffuse probes a single cubemap volume can now represent more localized bounce lighting.

Sky visibility allows light from the outside to bleed in.

LIGHTING FEATURES SUMMARY

- Area specular
- Sky hemisphere
- Local cubemaps
- Diffuse probes

To summarize lighting, we updated our deferred lights with area specular, and added an image based lighting pipeline, from global to local, sky hemisphere, local cubemaps, diffuse probes.

ARE WE DONE?

- New material model
- Lighting solutions supporting it
- Feature set fulfills our art goals

- Let's make art!



To fully summarize our Destiny 2 shading technology, we implemented a new material model and lighting solutions supporting that new model, this feature set fulfills the requirements and goals of our art direction.

Are we done? Now we have to make art.

TRANSLATING TO PRODUCTION

Nate Hawbaker

Graphics Technical Art Lead

BUNGIE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

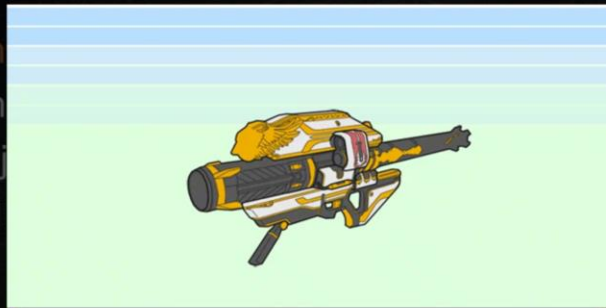
Hello, my name is Nate Hawbaker. I joined Bungie in 2011 and currently act as the Graphics Technical Art Lead.

Today I'm going to talk about how we integrated the mentioned changes to our technology, into an already-moving production environment, and some of the challenges we encountered along the way.

REQUIREMENTS

Dynamism and

- Dynamic time
- Dynamic obj
- Open World



... WHAT? ▼

variety and volume

complex shaders

~50k / world

count (50)

layer count

in combat

6 in non-combat

Evolution of a Live game

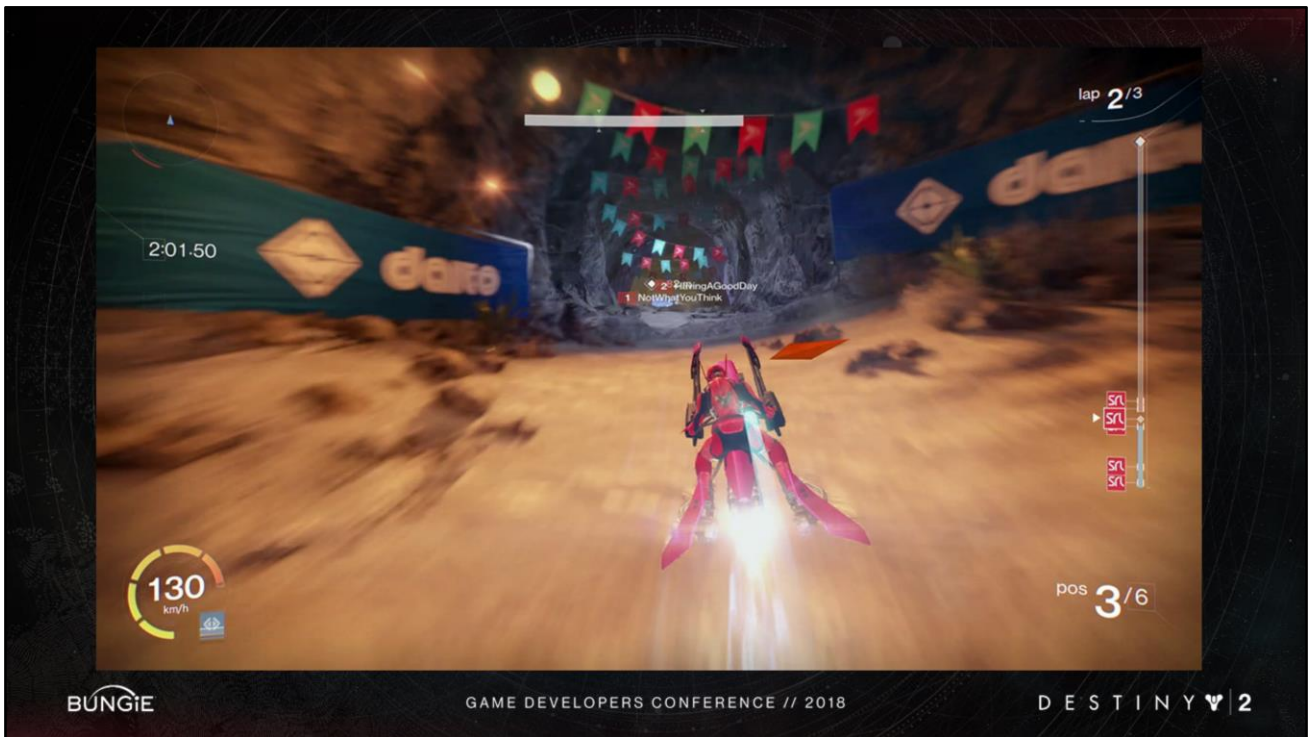
BUNGIE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

Earlier we saw these constraints, but there is a crucial entry missing from this list that informs our day-to-day in Graphics...

Evolution. Even after our game has shipped, it must be able to constantly change. Evolving with regular experience updates through patches and expansions.



By the end of the Original Destiny, our game had added locations in snowy mountain tops inhabited by wolves
miles-long Alien ships nestled in the rings of Saturn
these were then inhabited by enemies with their own lighting model
and naturally, we made a racing game.

Since we don't want our renderer to get in the way of the creative desires that produce the Live game of Destiny, our graphics must be able to SAFELY scale to experiences we haven't made yet.

FEATURE PARITY

- Opaque
- Layered materials
- Terrain
- Foliage
- Decals
- Transparency
- Lens flares
- Lights
- Screen effects
- Particles
- Player Gear



A large part of that is feature parity. Every one of the features Alexis described earlier is developed with support for all of its relevant feature-renderers.

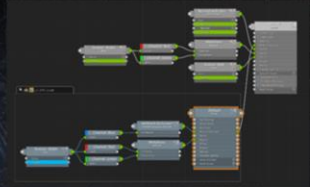
Terrain must support iridescence, foliage has to be able to support cloth fuzz, even decals are built to support things like transmission. You get the idea.

To simplify context switching, content creation and modularity, every authoring interface for these features is unified into a single system.

Engine Materials Lighting **Mat. Authoring** Mat. Modification Mat. Validation Gameplay

SHADER EDITOR

- Used to build **all** shader content
- TFX components contain CPU bytecode and HLSL code snippets
 - Authored by Engineering or Tech Art
 - Used for all non-artist-facing pixel shaders
- Supports real-time bytecode updates to engine
 - [Tatarchuk & Tchou 17]



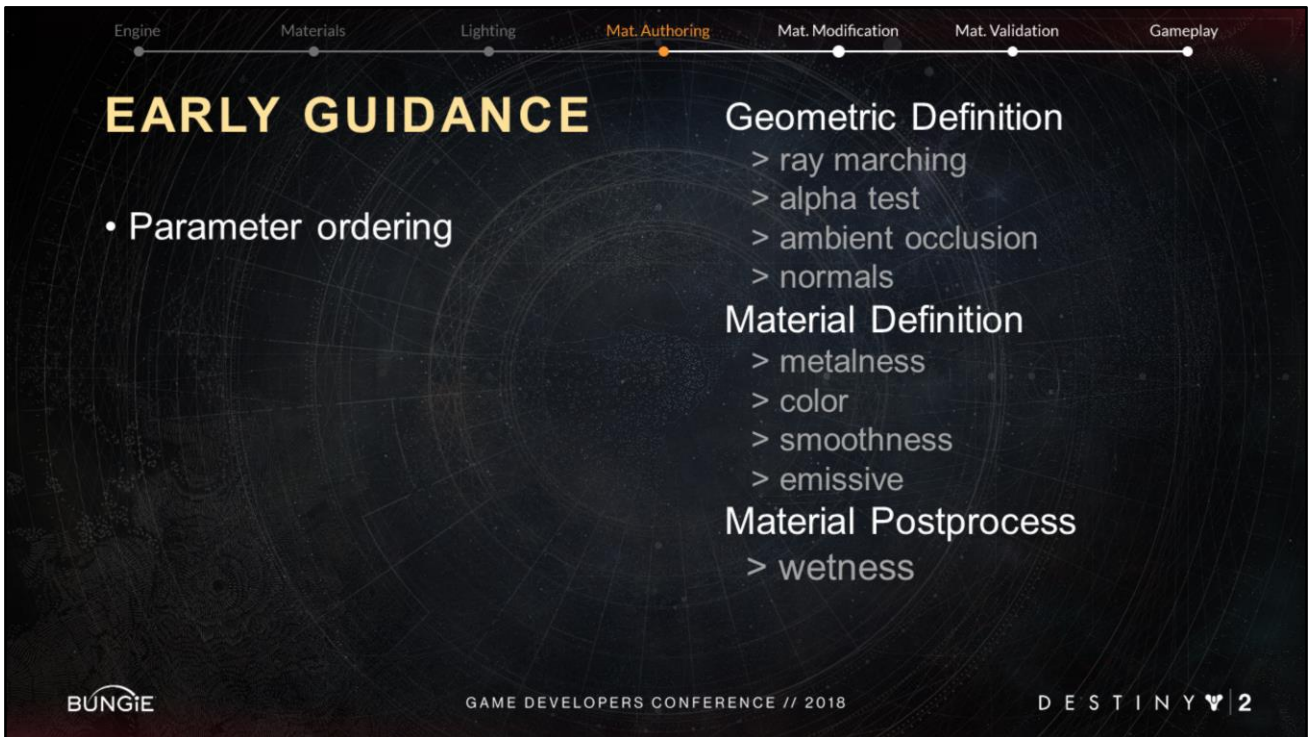
BUNGE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

That system is a node graph. At a high level this is where artists author all shader content in Destiny.

Each component contains CPU bytecode and HLSL functions, and all bytecode parameters can be updated in realtime to a running instance of the game

We went into technical detail about some of the inner workings of it at last year's GDC, and I'll have links at the end of this talk.

The reason that I wanted to start with the shader editor is because for many artists, this is the earliest point in their pipeline that we can provide measurable improvements in guidance.



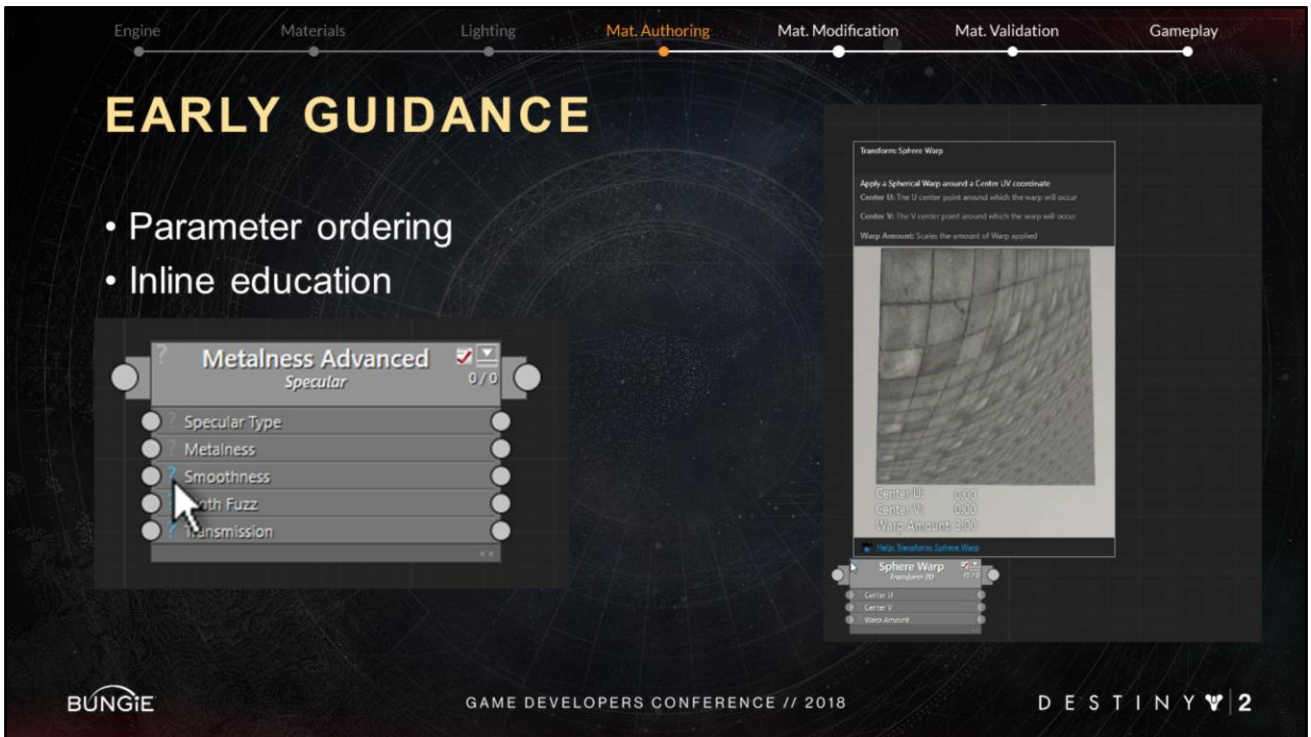
So to pick up where we left off with Alexis, we were just starting to roll PBR out to artists.

One of the things that can catch an artist off guard is that now there are both **right** and **wrong** choices when authoring surfaces. So how can we *guide* our artists towards making safe choices?

We start with a simple question: What order are parameters seen in? This may seem inconsequential, but done poorly and your artists may often find themselves revisiting parameters they've already authored.

Things like alpha test and ambient occlusion are fairly agnostic of your material. But a parameter like Metalness can create restrictions on what colours are valid, so it comes first. Before this we found artists authoring half-metal surfaces at a fairly alarming rate.

Simply considering the order that you expose parameters to your artists, can lead to better art.

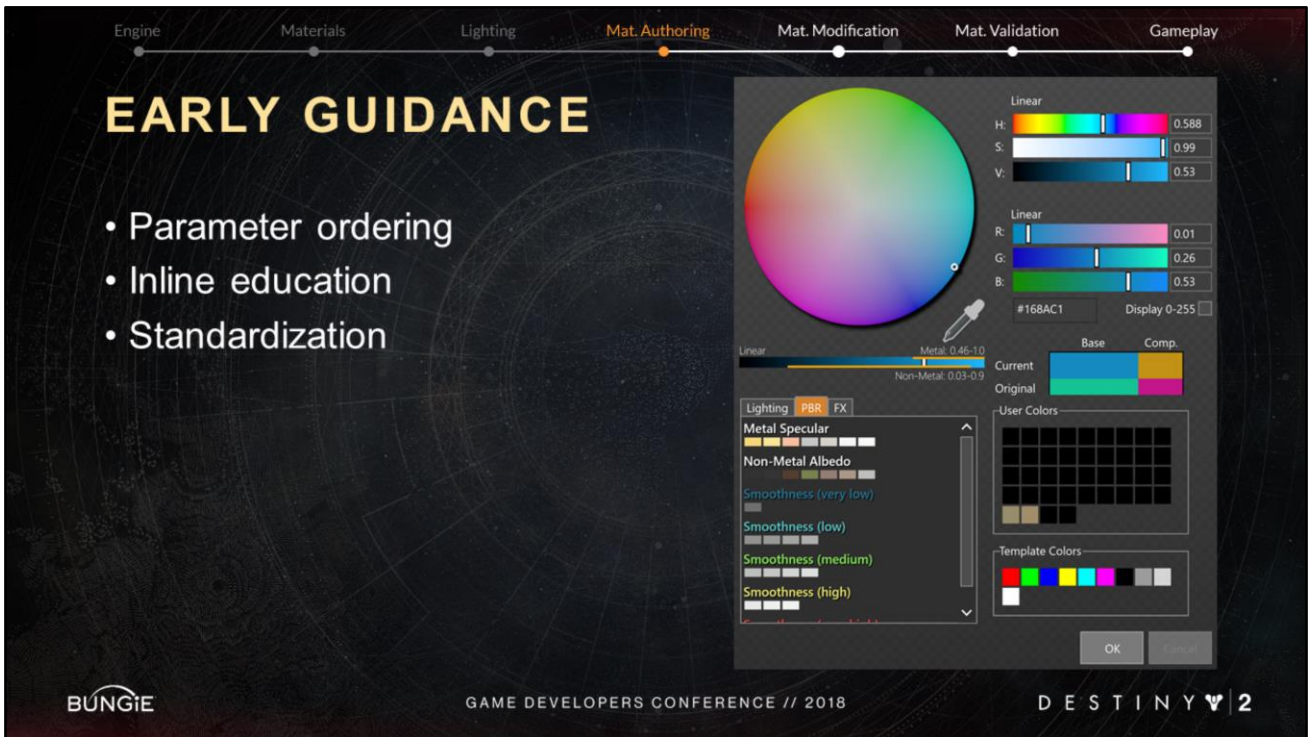


To help with education, each shader component and parameter can provide their own inline documentation. Anytime that an artist sees a blue question mark, something has been written by their coworkers.

If the question mark is grey, no information is available, and instead they're invited to help contribute to the effort.

Every ToolTip supports RichText, images, videos, and links to additional internal documentation resources.

This an example of a ToolTip for a Sphere Warp transform. You can see the video not only shows what it looks like, but provides feedback for what values are being used in the example.

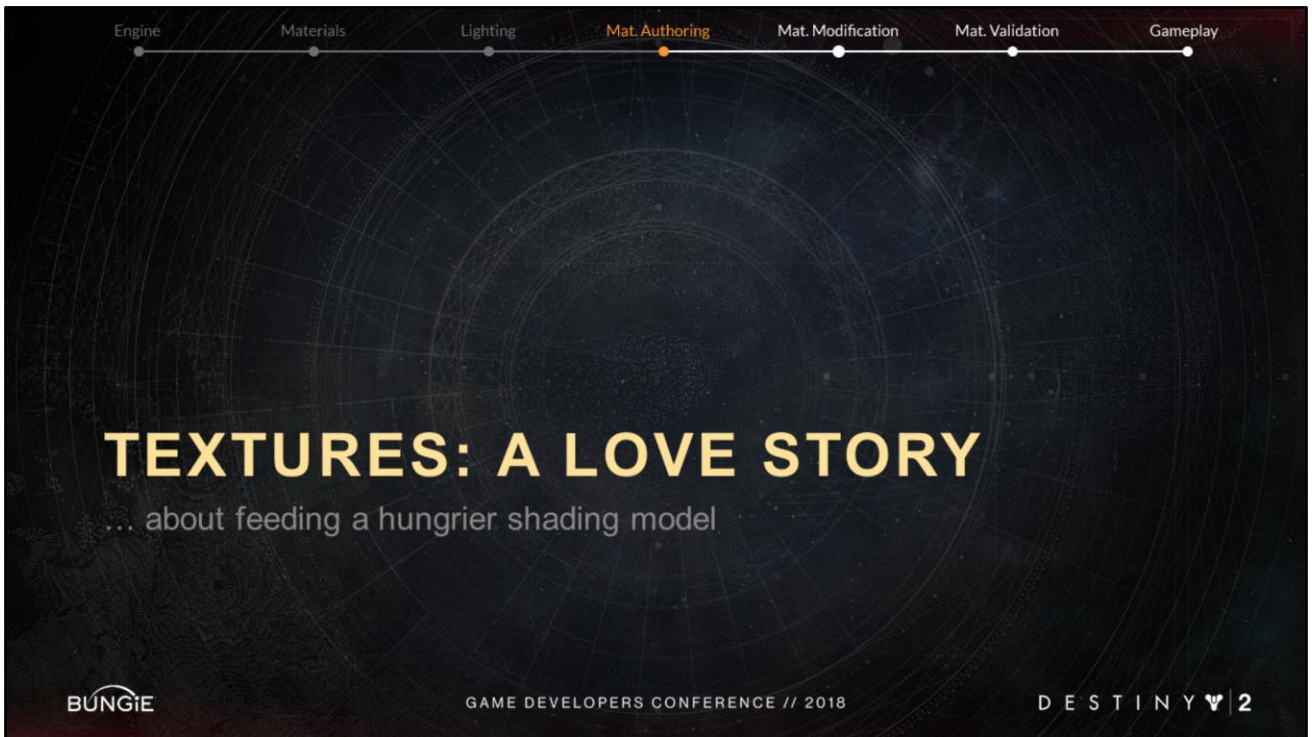


EARLY GUIDANCE

- Parameter ordering
- Inline education
- Standardization

And to help with choosing the right inputs that are both safe and standardized, our Colorpicker contains known PBR values for both colour and smoothness, as well as reminders of project guidelines for color brightness/metalness relationships.

But most inputs *aren't* constant colours, complex variation is still commonly provided by textures – and this begins our next section.



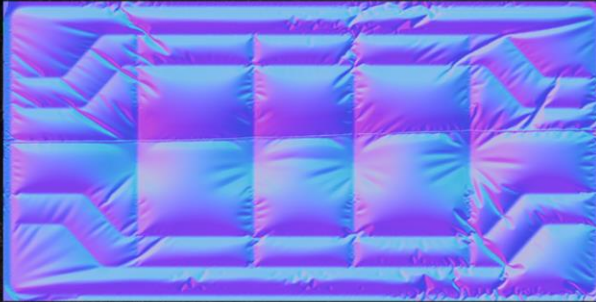
Let's talk about textures!

As material models grow more sophisticated, so must our optimization strategies. As Alexis mentioned earlier, sometimes this can be Gbuffer encoding optimization process, but other times the optimization is within the scope of the textures in your content.

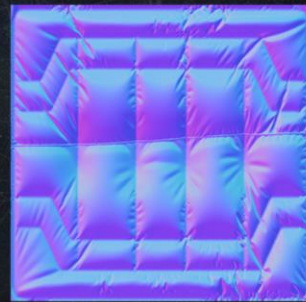
OPTIMIZATION: RESOLUTION

- BC1 compression

Source Resolution: 2048 x 1024



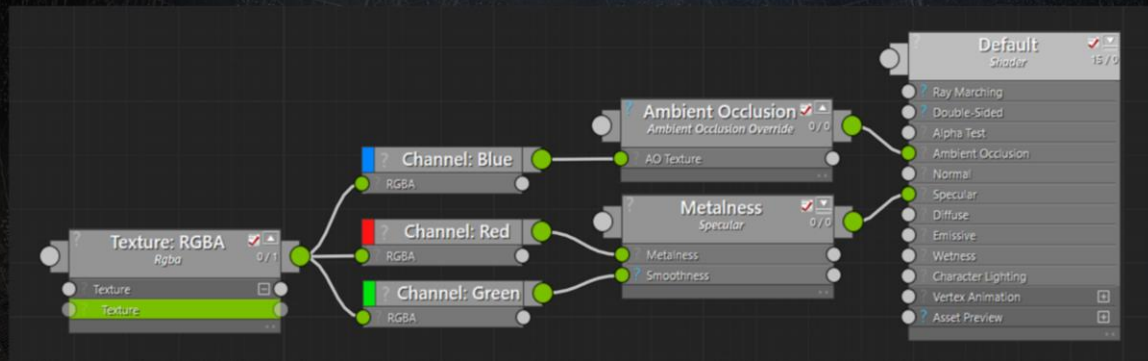
Compiled Resolution: 1024 x 1024



Sometimes this means revisiting compression schemes and reducing resolution. In an effort to reduce the streaming memory footprint of environment normal maps, we reduce to half-resolution on the longest axis, and compress in BC1.

OPTIMIZATION: STORAGE


- BC7 compression
- Metalness / Smoothness / Ambient Occlusion stack



More commonly this is stacking parameters across each channel of a texture, and opting for a friendlier, data-focused compression like BC7.

Engine Materials Lighting **Mat. Authoring** Mat. Modification Mat. Validation Gameplay

OPTIMIZATION: PARAMETER ENCODING



Color Change

Transparency

Transparency < 128
Color Change > 128

BUNGE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

But sometimes you need more than four channels. If your parameters are mutually-exclusive you can encode and decode them from a single channel.

In this example we're considering everything above 128 to be a color change mask, and everything below it to be transparency. Since a texel can't be both transparent and color changing, we have parameter mutual exclusivity.

TEXTURES? WHAT'S YOUR POINT?

- More PBR == more inputs
 - [Lagarde 17]
- Multi-layer BRDFs
 - [Drobot 17]
- NDF anisotropy shape control
 - [Ribardiere 17]
- Multiple scattering
 - [Heitz 16]
- Iridescence / Airy reflectance
 - [Belcour 17]

But why am I even talking texture storage in a talk about PBR?

Though PBR is about simplifying the job of an artist, it often requires more inputs to be defined than before. Sébastien Lagarde began conversations about future scalability challenges during his SIGGRAPH talk last year. As we look forward to the future of extending PBR models, there's a lot of awesome things on the horizon!

But all of these need inputs – and we, as an industry, are faced with tackling the challenges associating with storing them... .. but you don't get points for using more non-blendable look up textures 😊



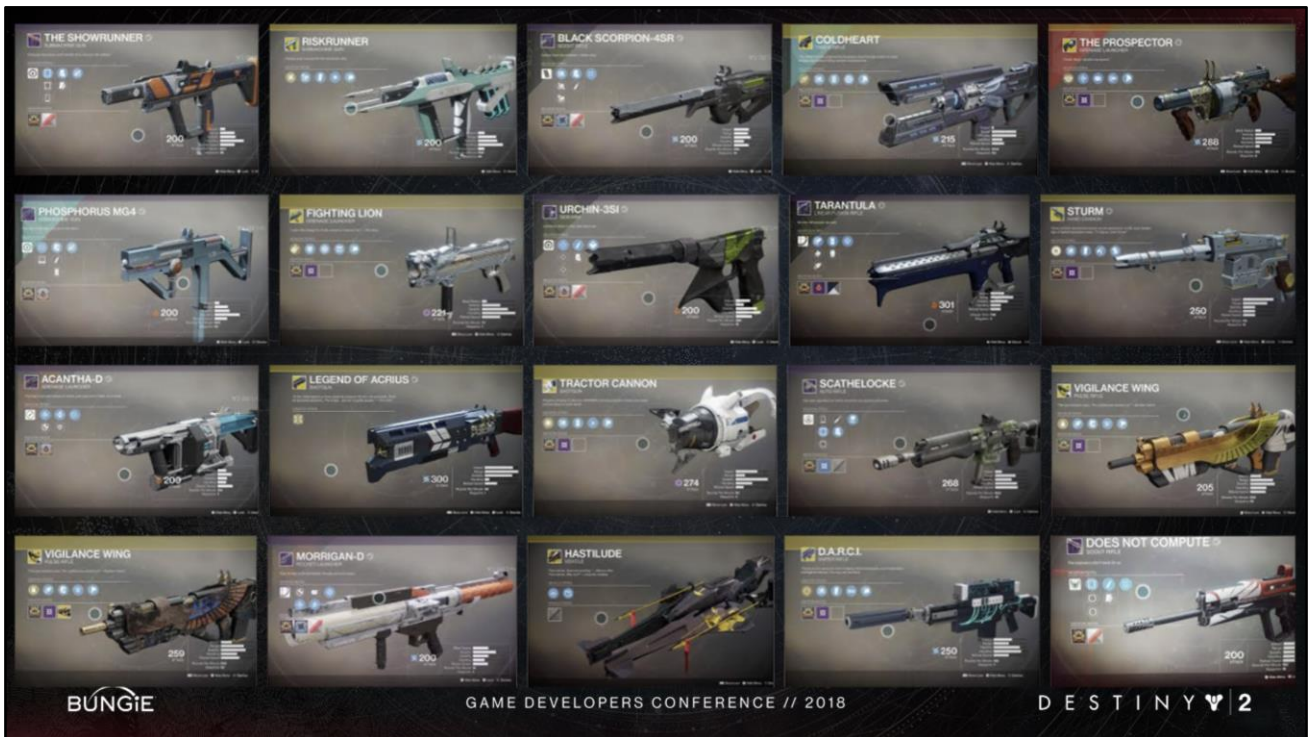
So let's look at a situation like this that we were already faced with.

Imagine you have a fixed function shader at texture sampler limits, and it can't change due to performance and scalability reasons. Even consider that *may* have just switched to a PBR model, creating the previously-mentioned need for an expanded set of material inputs.

This shader happened to be the one that applies to all player customized gear in our game... a game about getting exciting and **diverse** gear.



All characters



All weapons



All the ships, vehicles, player Ghost companions – it's a lot of content.

CONSTRAINTS

- One texture to fit 7 parameters in 4 channels
 - Sub channel stacking
 - MIPs can create unexpected parameter shifts
 - [Tatarchuk & Wang 14]

We went into detail about the player Gear system in a SIGGRAPH 2014 talk. Please check it out if you're interested in seeing how this texture fits in our ecosystem.

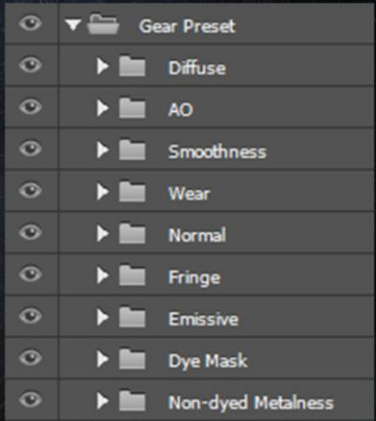
For now, let's review the Constraints of the problem space

We have to fit 7 parameters into 4 channels. This implies parameter encoding into individual channels, with considerations for both mutual exclusivity and MIPing. MIPing can cause premature parameter boundaries to be crossed based on angle or distance, so you **also** have to be conscious of the *order* that things are encoded, especially for concepts like emissive.

Engine Materials Lighting **Mat. Authoring** Mat. Modification Mat. Validation Gameplay

CONSTRAINTS

- One texture to fit 7 parameters in 4 channels
 - Sub channel stacking
 - MIPs can create unexpected parameter shifts
 - [Tatarchuk & Wang 14]



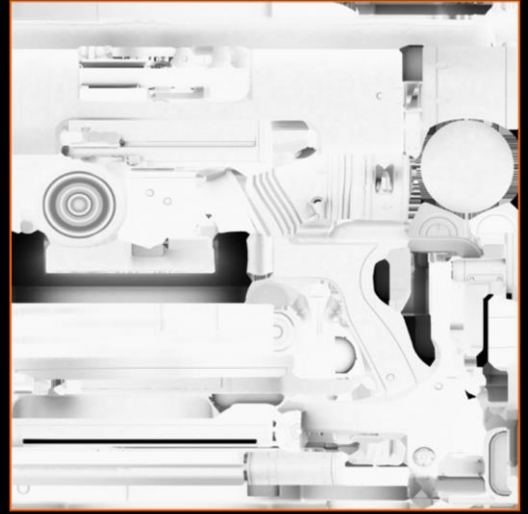
BUNGE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

It's worth mentioning that all of the encoding and channel storage that we're about to talk about is done automatically, artists paint simple 0-1 masks and click Export

0

255

R Ambient Occlusion



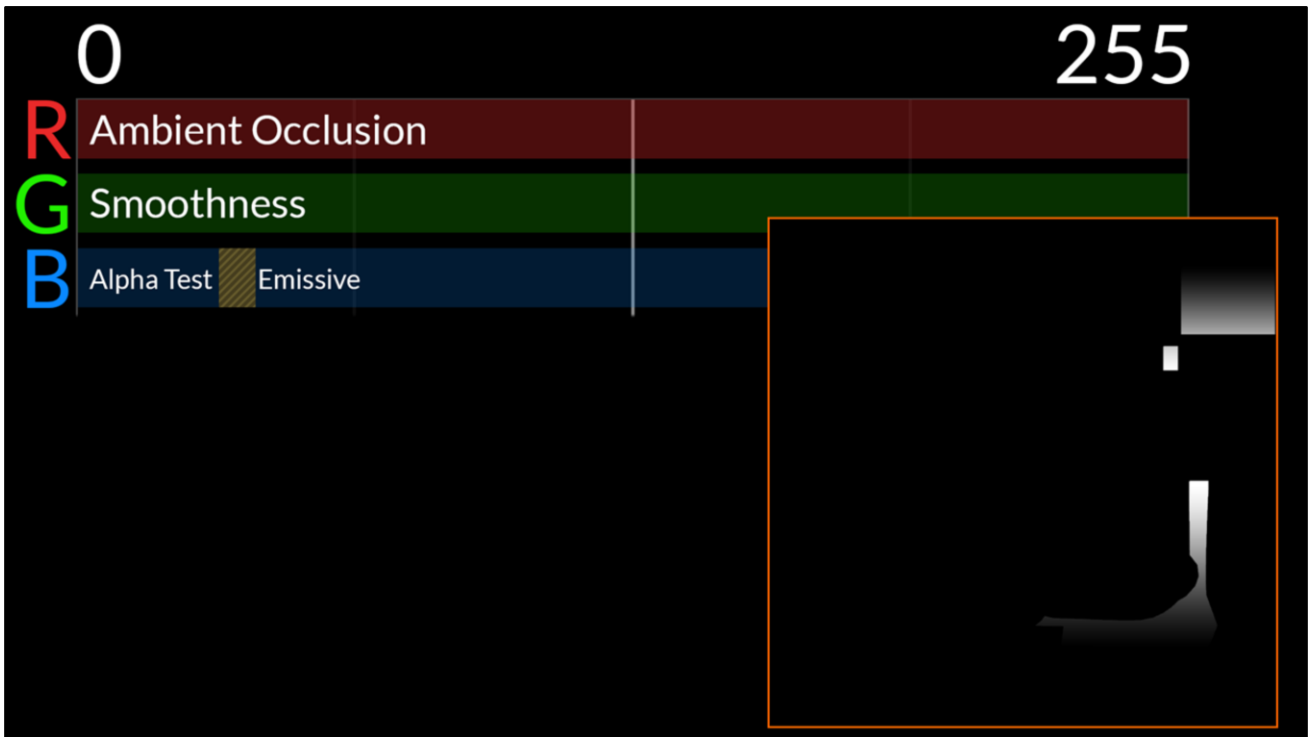
So let's start out with the status quo. Ambient occlusion is attenuating HDR lighting, it gets the full 255 range.



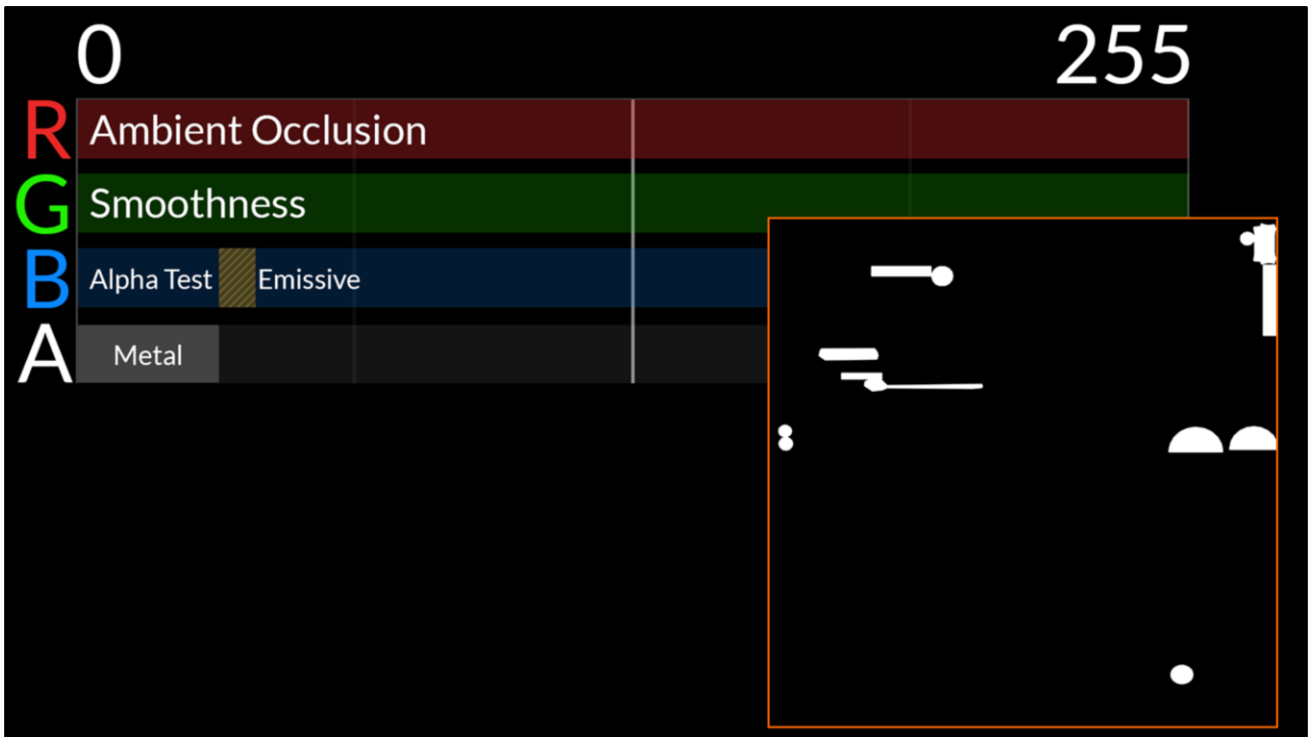
Smoothness can make or break your HDR specular highlights, it gets all 255 values too!



Now we can get creative. Some of our capes need alpha test to provide fringing on the bottom. We gave this 32 values so that we could animate across gradients for specific effects.



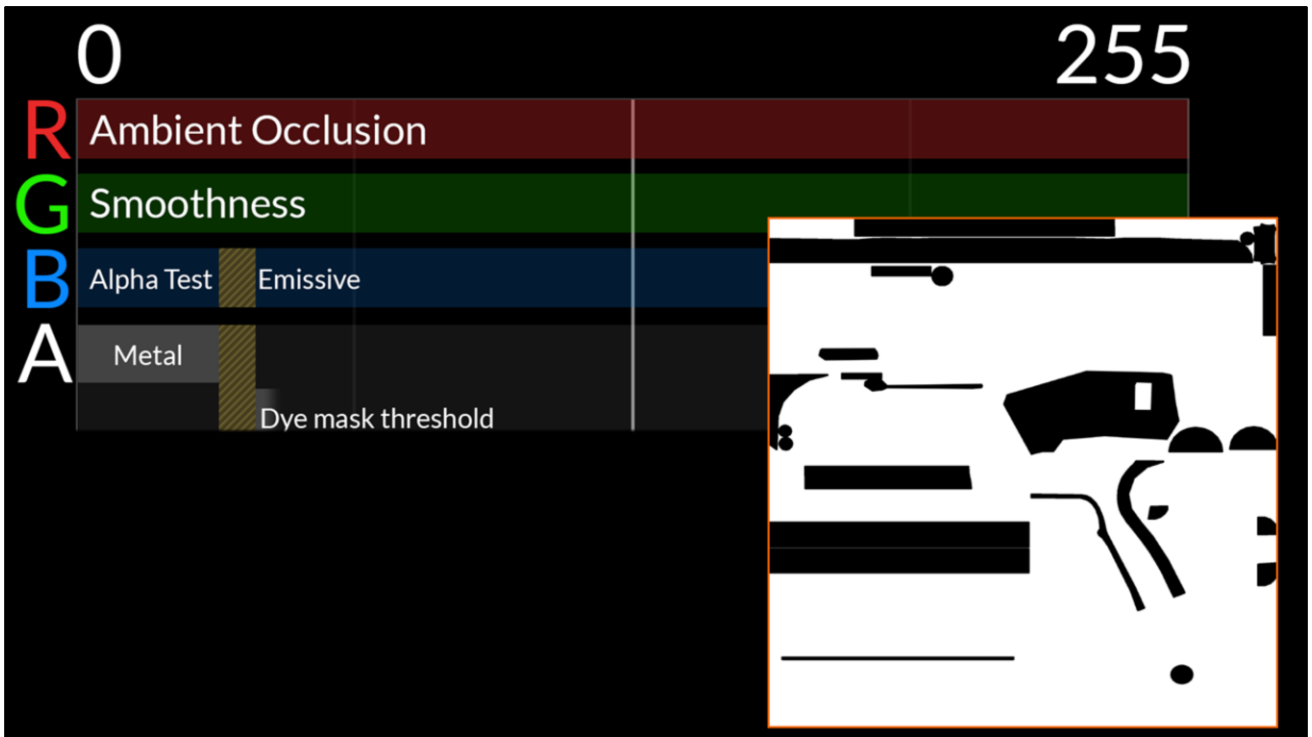
We pad by eight values and then give emissive 40 to 255. If something was alpha tested out, it can't be emissive, so we have mutual exclusivity.



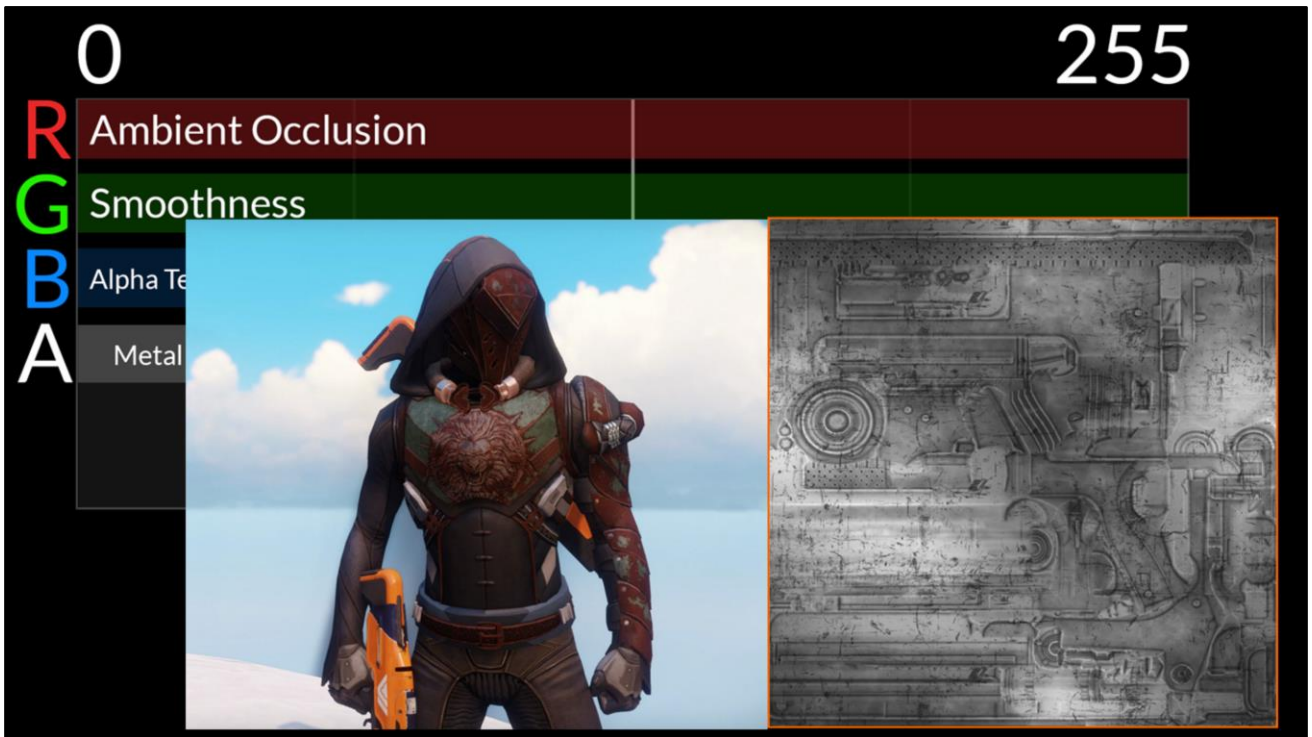
In the alpha channel we give metalness 32 values.

We agreed with artists that we could avoid large dielectric to conductive material transitions where stepping would be more apparent.

An important distinction is that this is actually something we call the “un-dyed” metalness. Dying is the term we use to describe areas that can be changed at runtime by player customization. These 32 values describe the metalness properties of the areas that players cannot customize.



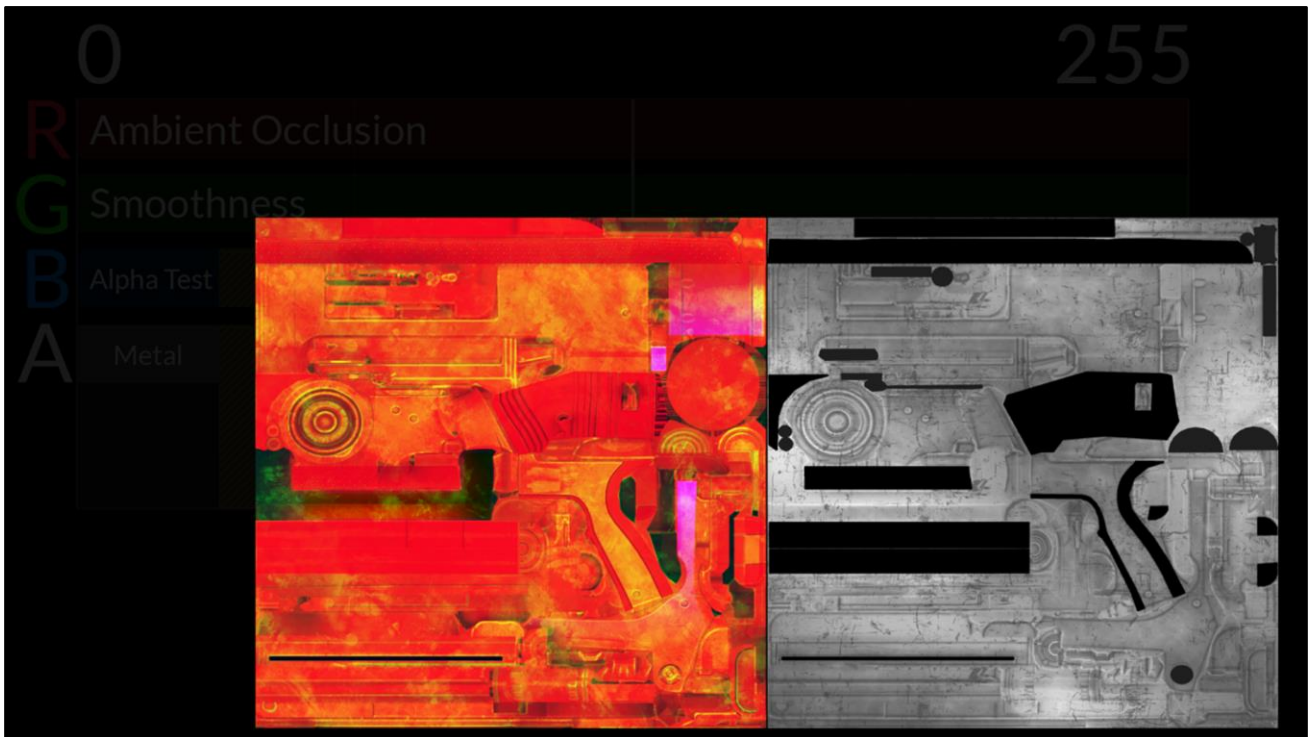
To define the area that the player can customize with dying at runtime, we apply a step function to this channel and consider anything **above** 40 to be customizable, once again using 8 values of padding.



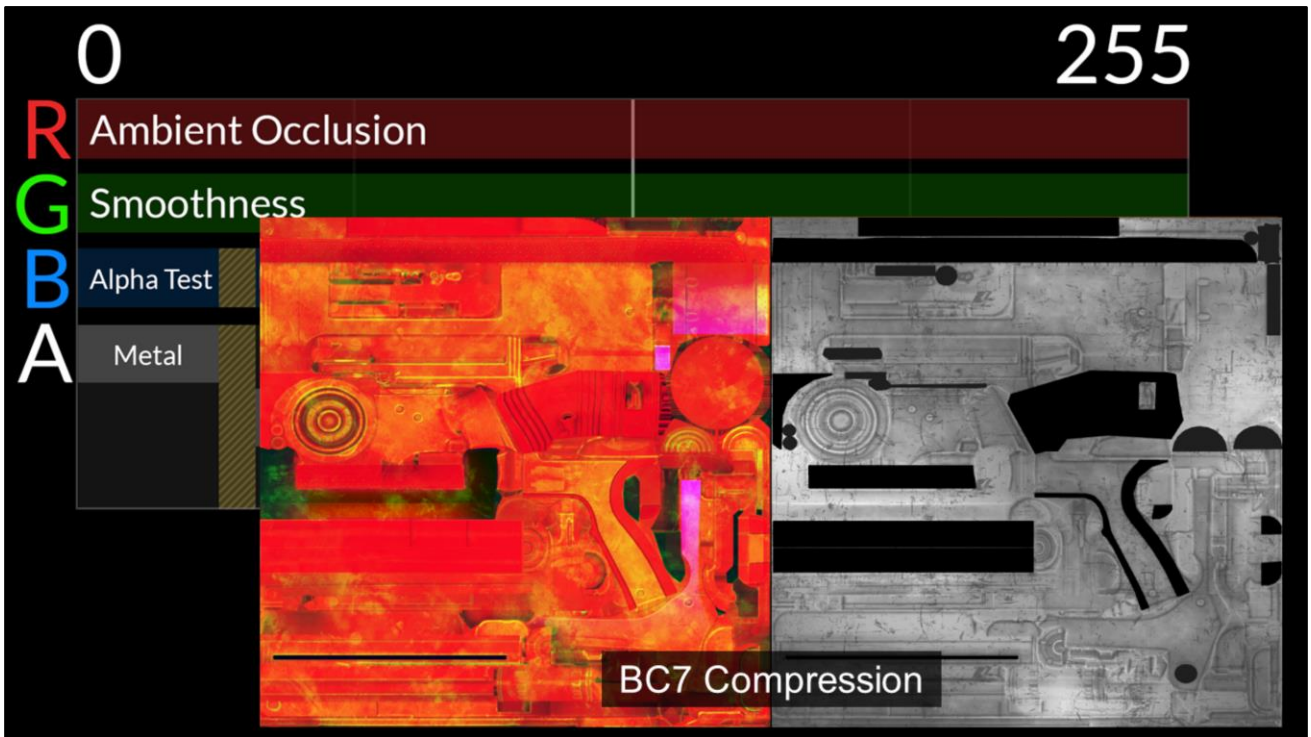
And last but not least, we store something called a wear mask between 48 and 255. Wear is a property of a dye.

If a player puts an awesome copper shader on their armor, we want it to wear or patina with oxidization in a physical plausible fashion.

Thankfully wear will only scuff away at the player customized areas, so we are able to get away with parameter mutual exclusivity again, packing our final 7th parameter.



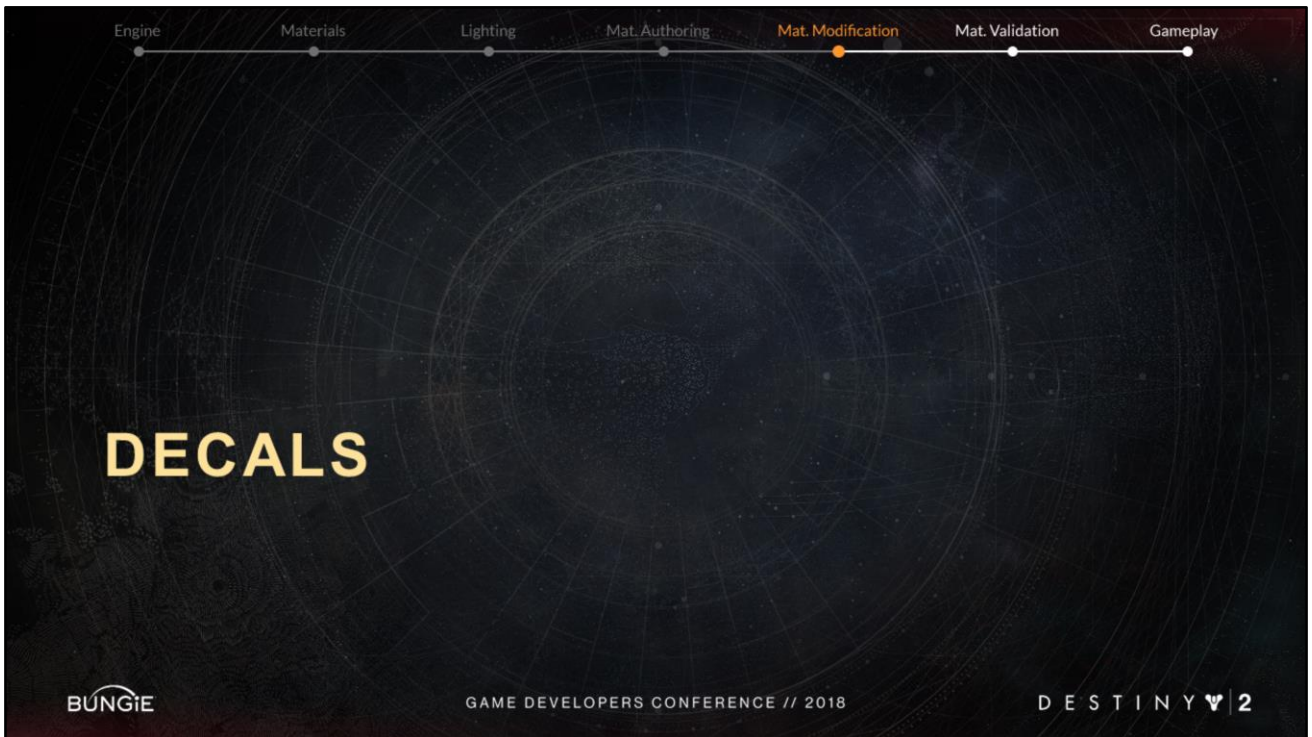
This is what it looks like once it's all packed together.
And again, artists never have to interact with this. they author SIMPLIFIED 0-1 masks in Photoshop, and we composite everything for them on Export.



We used BC7 compression on this texture we didn't encounter that many issues. The biggest change I would make if I had a time machine be to give Metalness more range and swap with emissive.

We were concerned about the HDR-nature of emissive showing obvious banding, but content adds so much high frequency noise to the signal that it effectively dithers for us. Metalness is shifting HDR lighting ratios heavily and should have been provided more room to breathe.

If you find yourself in similar constraints as us, know that it *is possible* to sustain this path of parameter storage. It just requires a bit of creativity and collaboration with your artists – and a working agreement can go a long way.

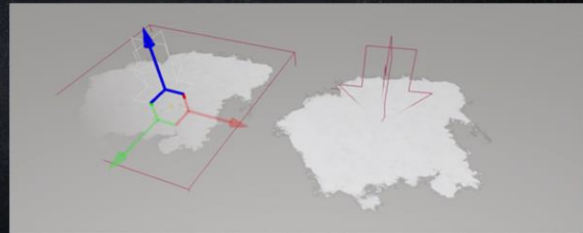
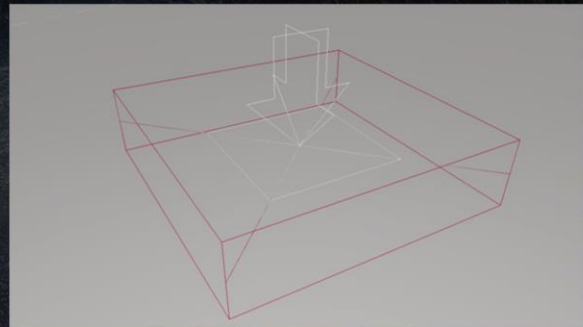


Now that we've fed all of our parameters and rendered our Gbuffer, let's talk about ways to change it!

The first pathway of doing that is decals.

DECALS

- Deferred projection volumes
- Placed via World Editor

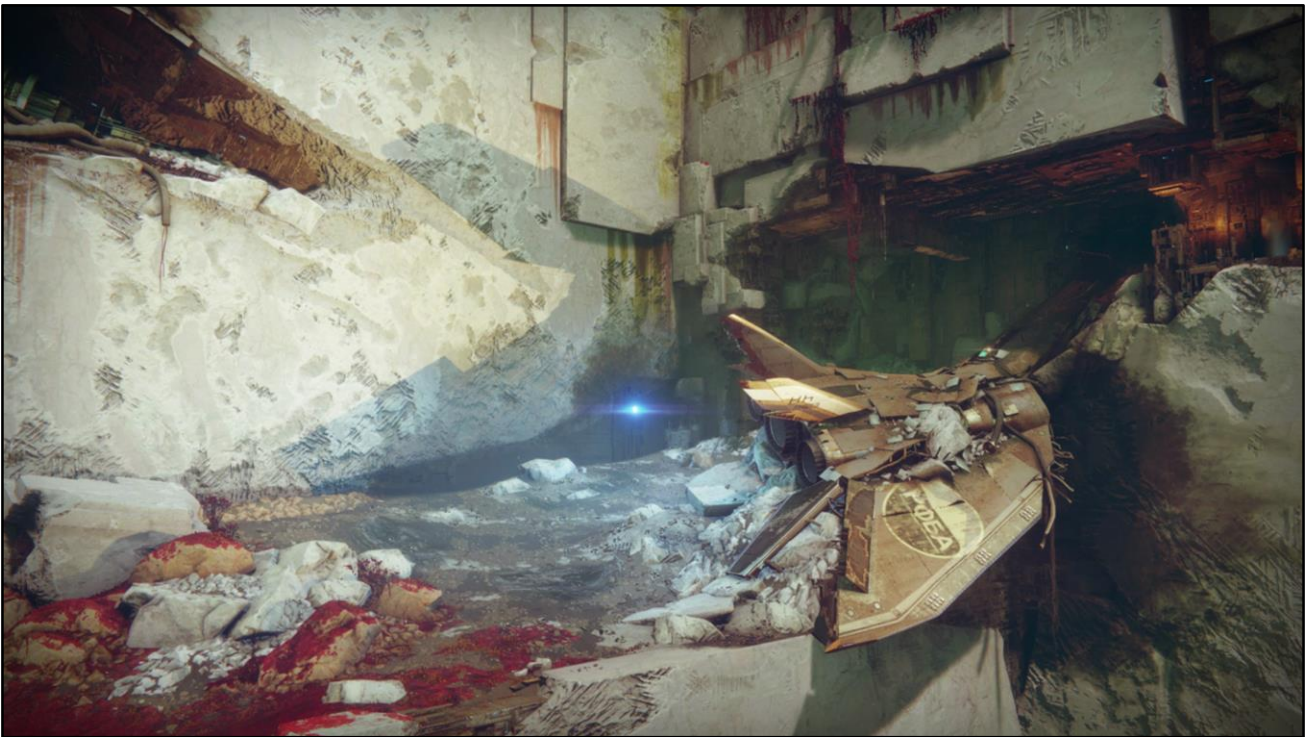


Decals in Destiny 2 are rendered as projection volumes. They're placed with our world editor. Each placement instance contains a shader reference. Pretty straight forward stuff.

But let's look at an example from Destiny 2.

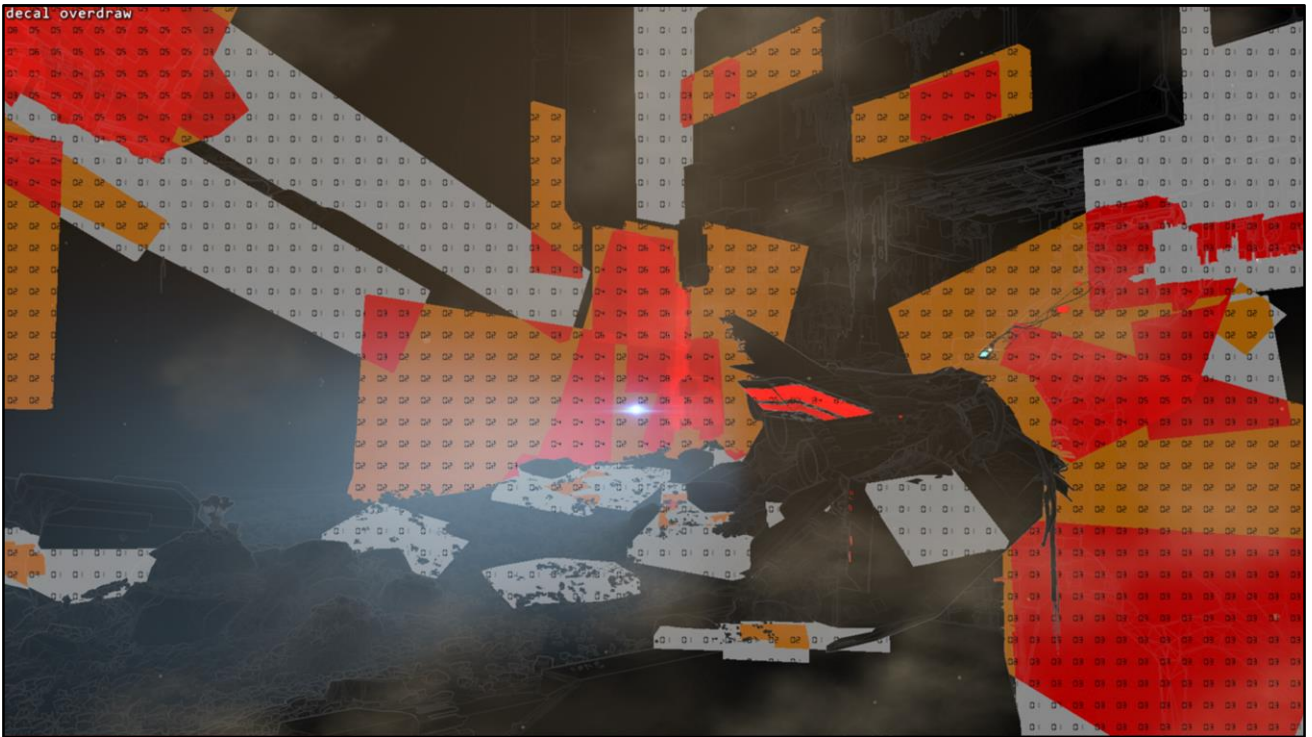


This is a destination called Nessus. We're going to be looking more closely at the area in the center of the screen.



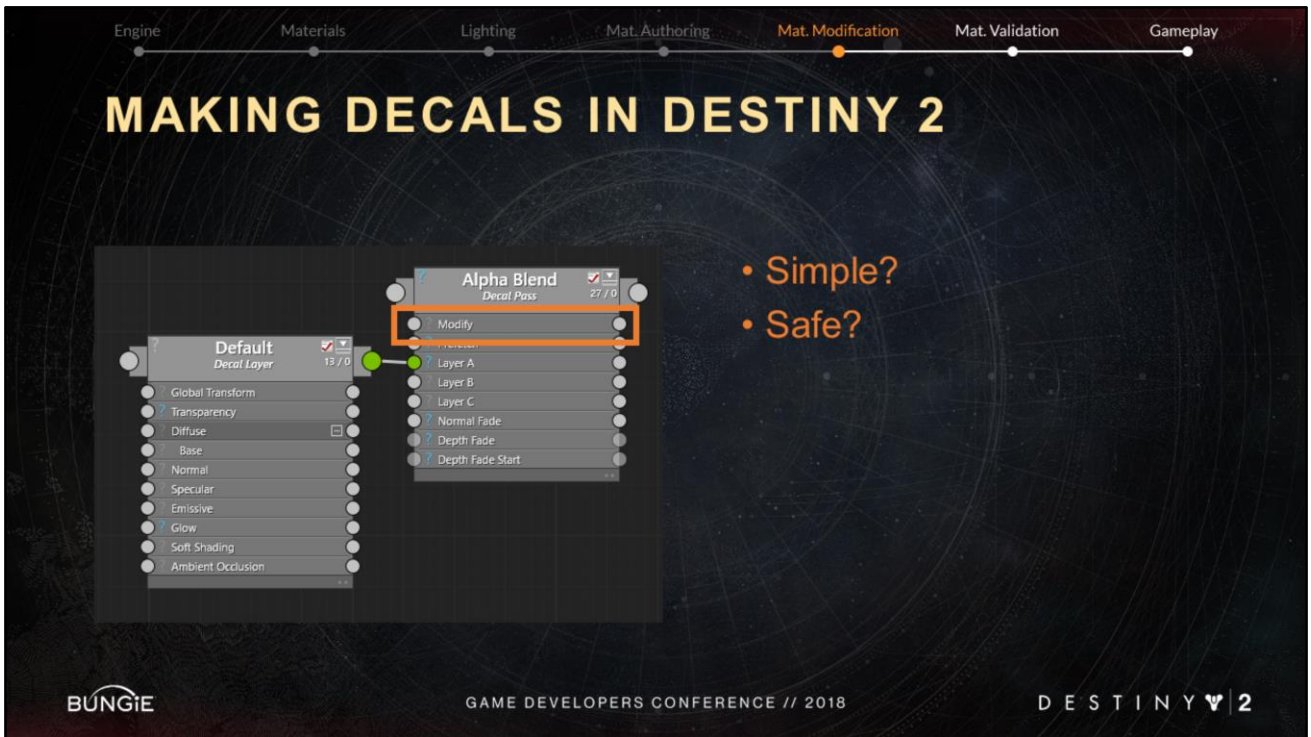
If I toggle decals on and off you will notice that our artists use them not just to place signs, but to actually paint features into the frame. Often this is to provide localized entropy without needing to create bespoke assets. However, this creates a demand for a wide range of content types that the shader system has to support.





This is a debug view artists use to identify overdraw. In a few hot spots you can see we're hitting up to 3-6x overdraw! With layer compositing to that degree, it was crucial for us to extensively iterate on the decal material ecosystem for both usability and safety.

Though we also support flavours of additive and alpha blended transparent decals, today I'll only be talking about deferred Gbuffer assets.



- Simple?
- Safe?

Making decal shaders is pretty straight forward. You have the familiar node graph setup with a material definition input.

But with an additional parameter called 'Modify' that sets a hardware write mask. This is the basic job of a Gbuffer decal. After everything has been rendered, how can you selectively blend different channels of your Gbuffer to produce new materials.

And this is where we start our next journey through production...

How do you describe a write mask to an artist, while still fulfilling our mandates of being both Simple and Safe?

Engine Materials Lighting Mat. Authoring **Mat. Modification** Mat. Validation Gameplay

'MODIFY' ATTEMPT #2

Albedo	t0rgb
Albedo, Normal, Smoothness	t0rgb_t1rgb
Albedo, Normal, Smoothness, AO	t0rgb_t1rgb_t2g
Albedo, Smoothness, Metalness	t0rgb_t1rgb_t2r
Albedo, Emissive	
Albedo, Metalness	
Smoothness, Normal	
Metalness, Smoothness, Normal, AO	
Metalness, Smoothness	
Smoothness, Normal, AO	t1rgb_t2g
Metalness	t2r
Emissive	t2g
Albedo, Smoothness	t0rgb_t1rgb
Smoothness	t1rgb
Normal, AO	t2rgb_t2g

How did we get here?

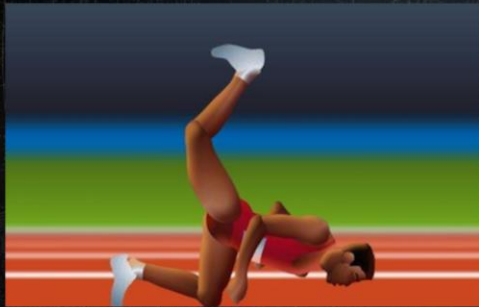
BUNGIE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

As missing combinations were encountered, we further filled them out based on requests, seeking to empower artists.

What we didn't realize was that we were creating an even bigger challenge for ourselves. The interface and code were growing even more unmaintainable. But worst of all, we realized we were hardly fulfilling our mandates of a SIMPLE system.
How did we get here?

THINGS ARTISTS DON'T (AND SHOULDN'T) CARE ABOUT

- Gbuffer layout



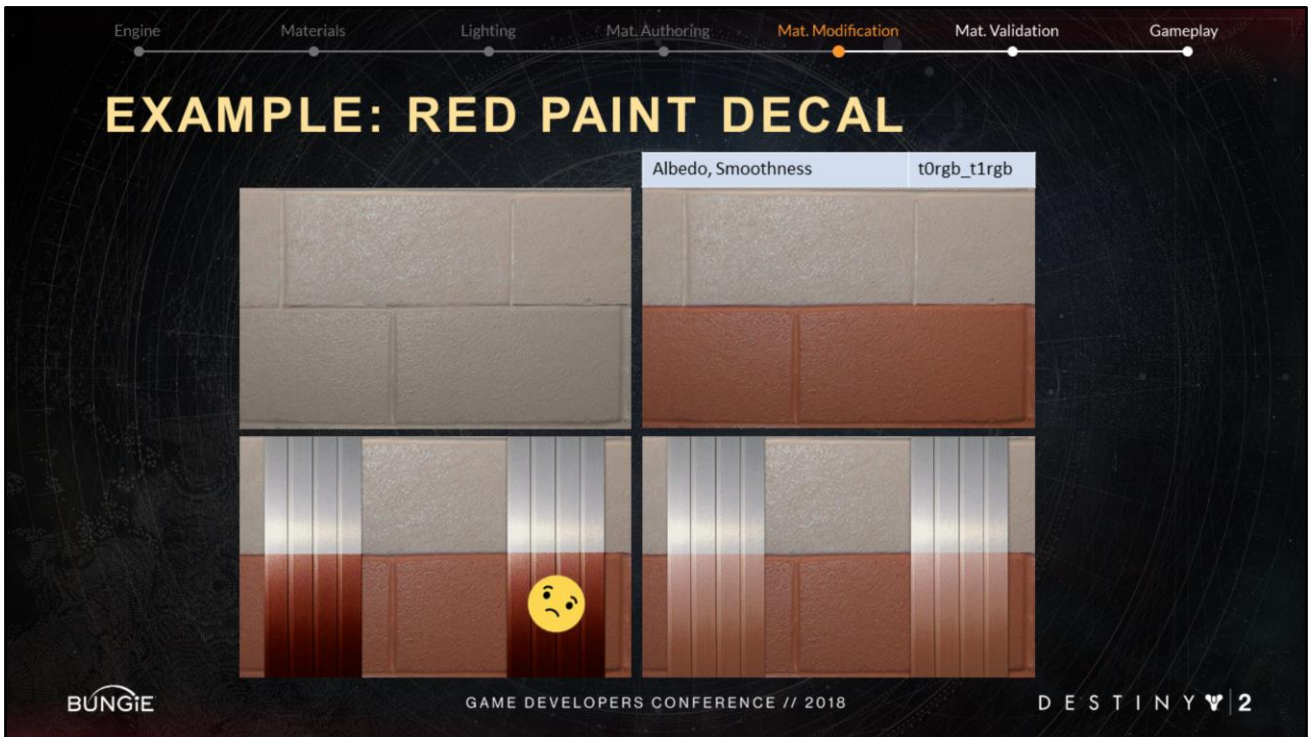
[Bennett Foddy] QWOP

Albedo	t0rgb
Albedo, Normal, Smoothness	t0rgb_t1rgb
Albedo, Normal, Smoothness, AO	t0rgb_t1rgb_t2g
Albedo, Smoothness, Metalness	t0rgb_t1rgb_t2r
Albedo, Emissive	t0rgb_t2g
Albedo, Metalness	t0rgb_t2r
Smoothness, Normal	t1rgb
Metalness, Smoothness, Normal, AO	t1rgb_t2rg
Metalness, Smoothness	t1rgb_t2r
Smoothness, Normal, AO	t1rgb_t2g
Metalness	t2r
Emissive	t2g
Albedo, Smoothness	t0rgb_t1rgb
Smoothness	t1rgb
Normal, AO	t1rgb_t2g

First, we asked artists to directly interact with something they don't, and shouldn't care about. A Gbuffer layout... .. I promise this list is entirely comprehensive. And yet, we were asking our artists to directly interface with it through **this**, *and* make the right choices, this was neither safe nor simple and even more importantly, we broke our QWOP metaphor about giving too much control, we were faceplanting.

Exposing artists to these ala carte channel writing concepts had an unexpected result, not only was it confusing, it encouraged them to make decisions backwards. Deciding what they want to **change**, rather than what they want to **preserve**.

Let me show you an example of what I mean because this is getting pretty abstract.



An artist wants to cover part of this wall with red paint.

Noting that they want the surface to be slightly smooth and red, they logically select Albedo and Smoothness from the Modify parameter list.

It looks good, done!

But then later, someone else adds metal elements to the original wall shader.

But because we didn't write to (and by extension preserve) the metalness buffer, we now have incoherent distributions of metallic paint poking through.

Correctly done, the paint decal should have provided an entire material definition in its writemask, including a metalness of zero.

DECALS: WHAT IS YOUR PURPOSE?

- What are we trying to achieve?



So let's take a moment to consider, what are we trying to achieve? Every technical possibility (apparently) didn't turn out to be our goal.

In fact, categorically things break down much simpler.


Thin things: Paint over concrete flooring and even *actual* 'decals'. The key is that these all preserve the underlying geometric representation while only changing material representation.

Total coverage: These are features like metal signage. These are decals that replace both the material AND geometric representation of the underlying surface.

Engine Materials Lighting Mat. Authoring **Mat. Modification** Mat. Validation Gameplay

SIMPLIFY AND ABSTRACT

Albedo	t0rgb
Albedo, Normal, Smoothness	t0rgb_t1rgb
Albedo, Normal, Smoothness, AO	t0rgb_t1rgb_t2g
Albedo, Smoothness, Metalness	t0rgb_t1rgb_t2r
Albedo, Emissive	t0rgb_t2g
Albedo, Metalness	t0rgb_t2r
Smoothness, Normal	t1rgb
Metalness, Smoothness, Normal, AO	t1rgb_t2rg
Metalness, Smoothness	t1rgb_t2r
Smoothness, Normal, AO	t1rgb_t2g
Metalness	t2r
Emissive	t2g
Albedo, Smoothness	t0rgb_t1rgb
Smoothness	t1rgb
Normal, AO	t1rgb_t2g



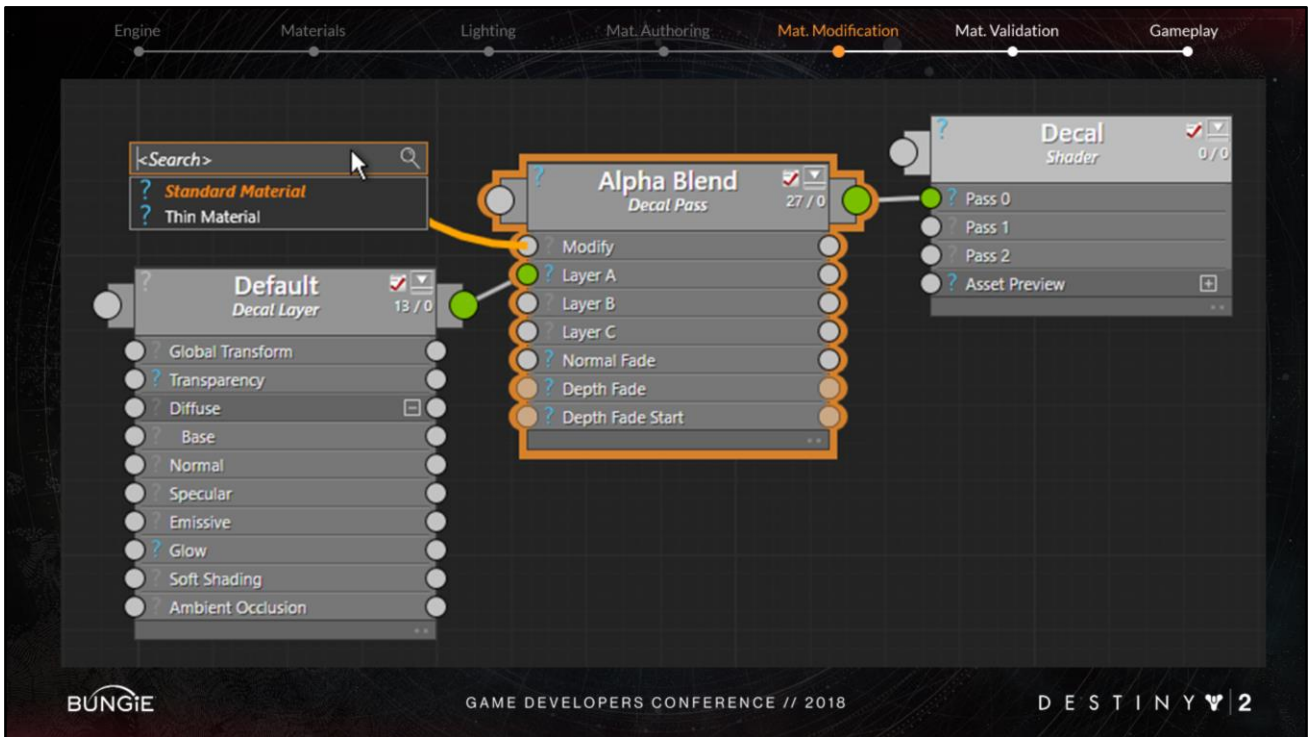
Thin Material	t0rgb_t1rgb_t2b
Standard Material	t0rgb_t1rgb_t2rgb

BUNGE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

So we simplified and abstracted. 15 Gbuffer write masks became 2 choices. "Thin" and "Standard" materials.

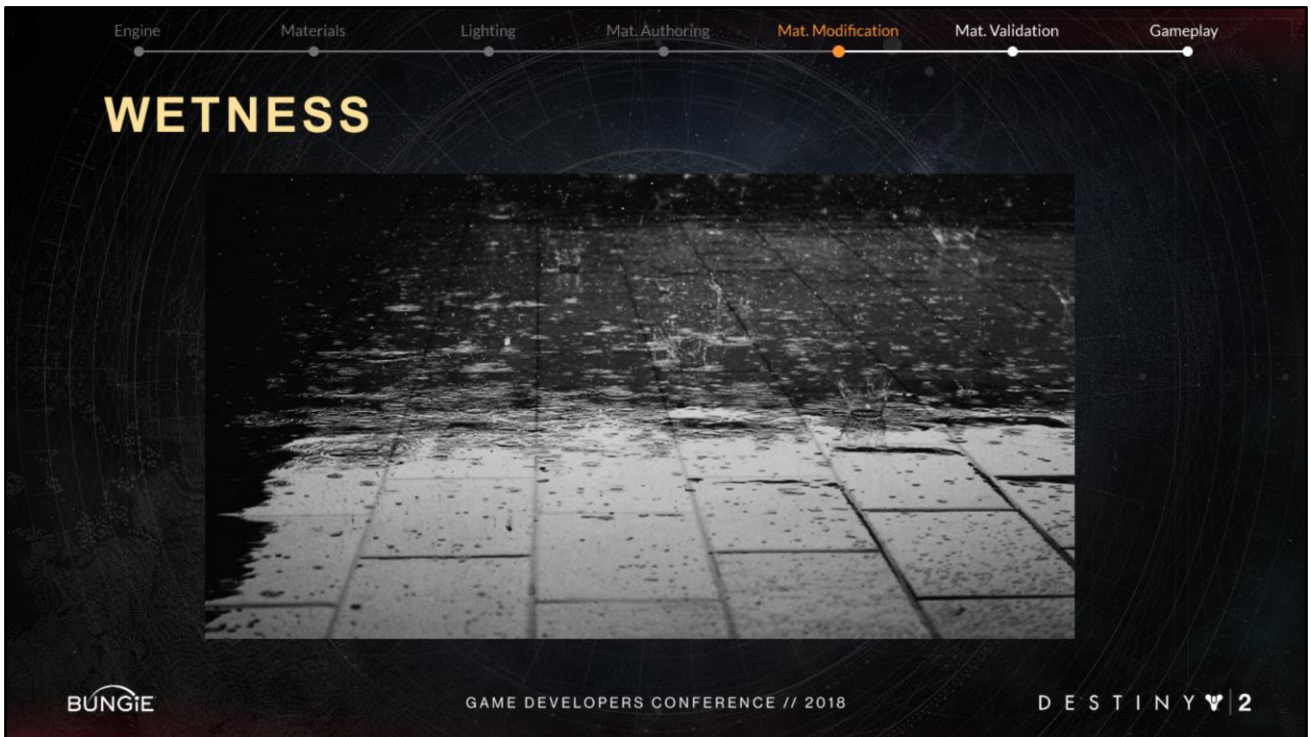
This provided both a safe and simple decal system through a layer of abstraction. It was now effectively impossible to forget to write to supporting Gbuffer channels that are so crucial to maintaining a healthy PBR ecosystem. Now, all our red paint example needs to do is use Thin Material.

This also provided a more subtle benefit of providing artists with learnable terms that can enter their vocabulary, making it simpler to provide accurate feedback and direction to each other.

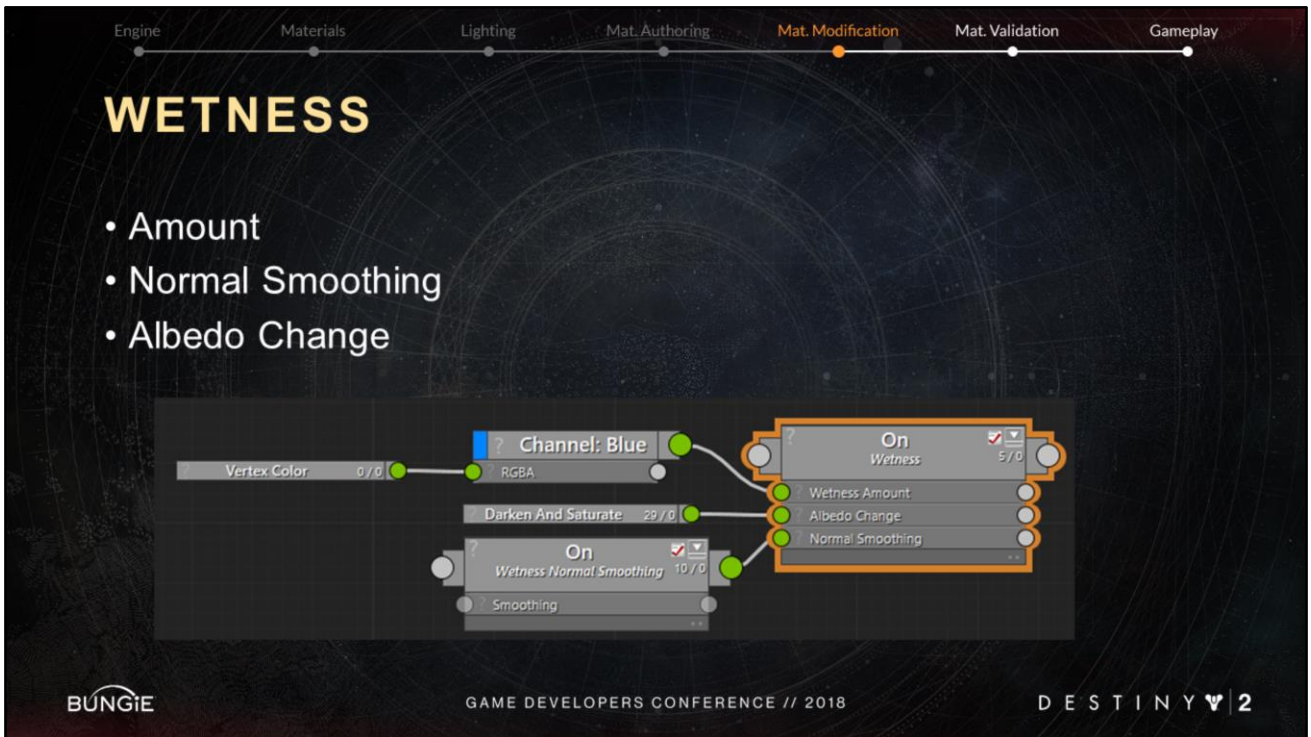


And to bring it all together, this is what it looks like in practice. Multiple decal passes are defined on the root node, intra-pass material layers are defined in the next node, and the greatly-simplified Modify parameter can be found at the top.

There were a few edge cases we opted to support by the end of the project. Geometric only changes for denting, and albedo darkening for puddles... Interestingly... if you make it easy for artists to create puddle decals, you may find yourself coming across puddle decals the size of a room.



That told us that our artists clearly had a desire to make surfaces feel convincingly wet on a large scale, but PBR inputs alone aren't not enough to simplify this. Because the current generation of hardware provides more ALU breathing room, we built a simple wetness parameterization into a new shader component.



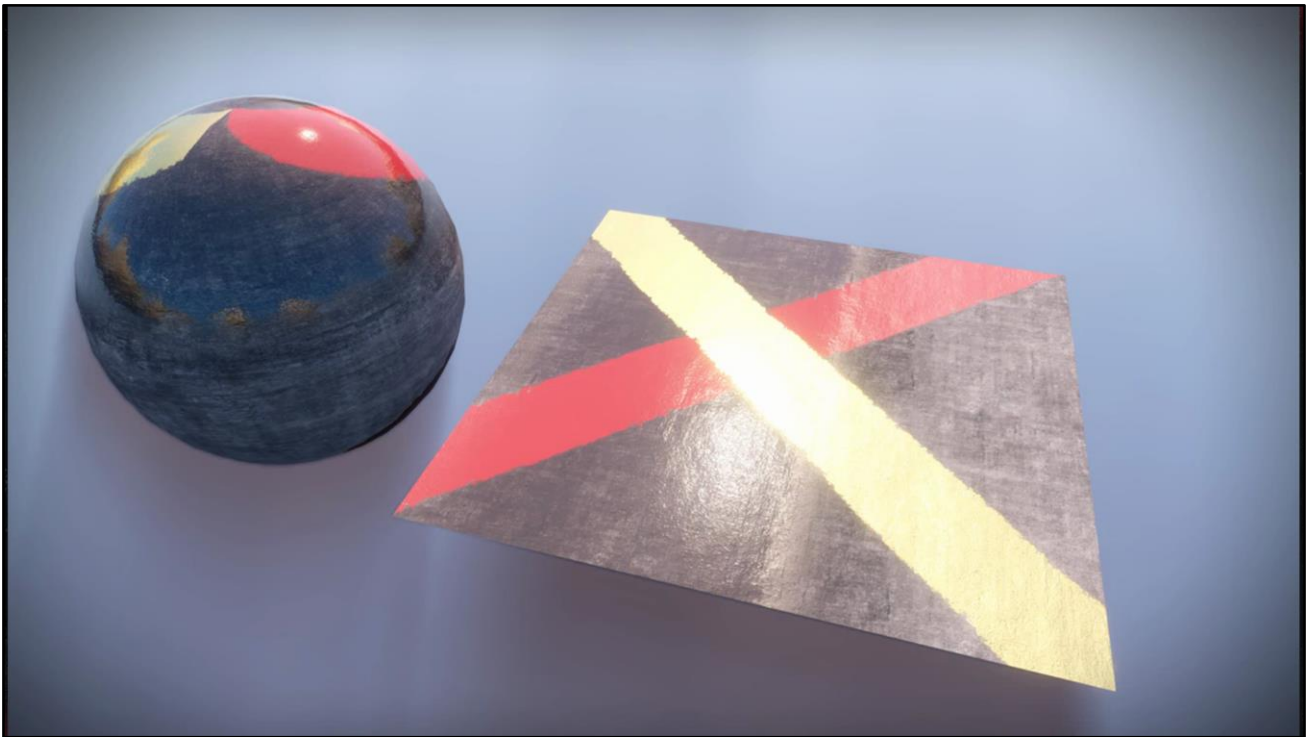
In the component, parameters are kept simplified and self-sanitize out of range values. Wetness amount is fairly straight forward. From 0-1 how much water is present on the surface.

Normal smoothing is a lerp to the geometric normal based on water saturation covering micro surface detail.

Albedo change is for modeling total internal reflection

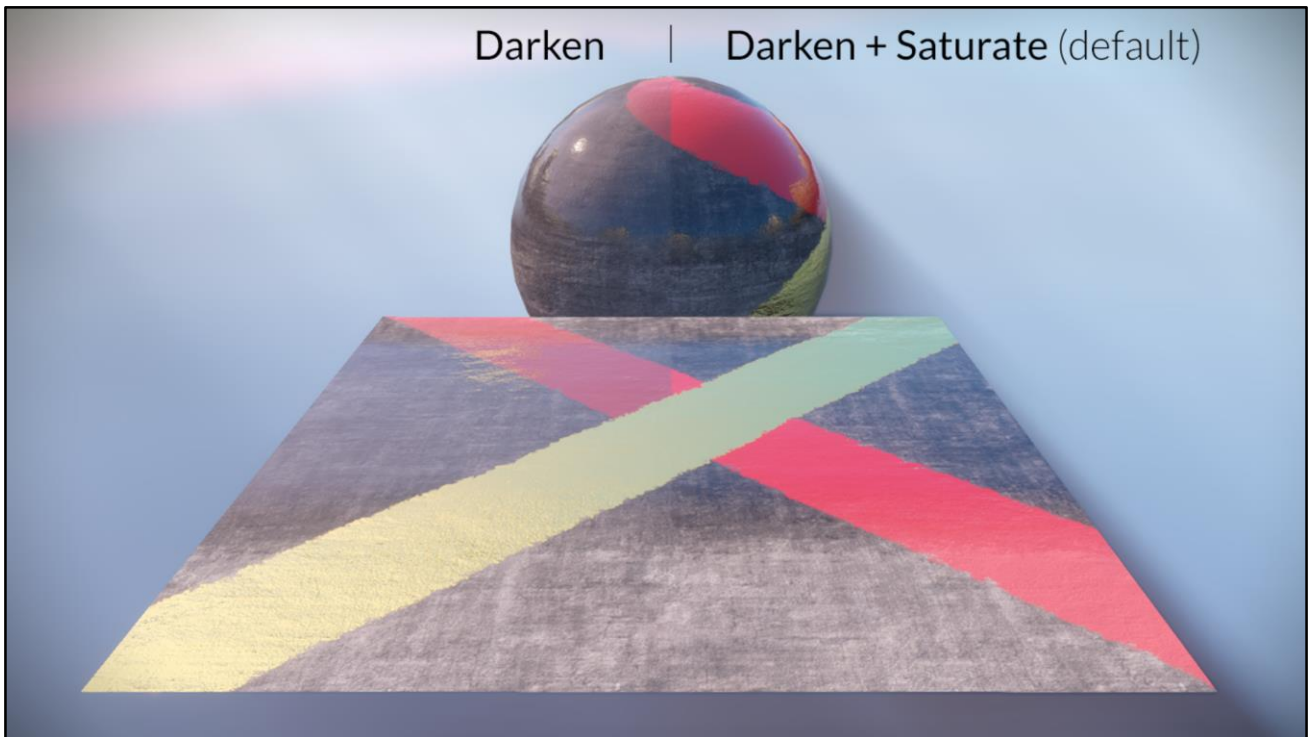
And to SIMPLIFY, porosity is implicit. We assume that metallic and smooth materials are largely 'sealed' surfaces and thus are non-porous.

Porosity will be at its highest on partially-retroreflective rough surfaces like clay.



This is an example of a wet gradient passing over a surface. You can notice that the gold metal behaves as a non-porous surface.

And the porous surfaces remain darkened for slightly longer than when the initial water evaporates from the surface.



And this is a side by side comparison of the Simple and Standard options for total internal reflection.

The more spectrally faithful version on the right is our current default behavior, and a simpler attenuate-only option can be seen on the left.

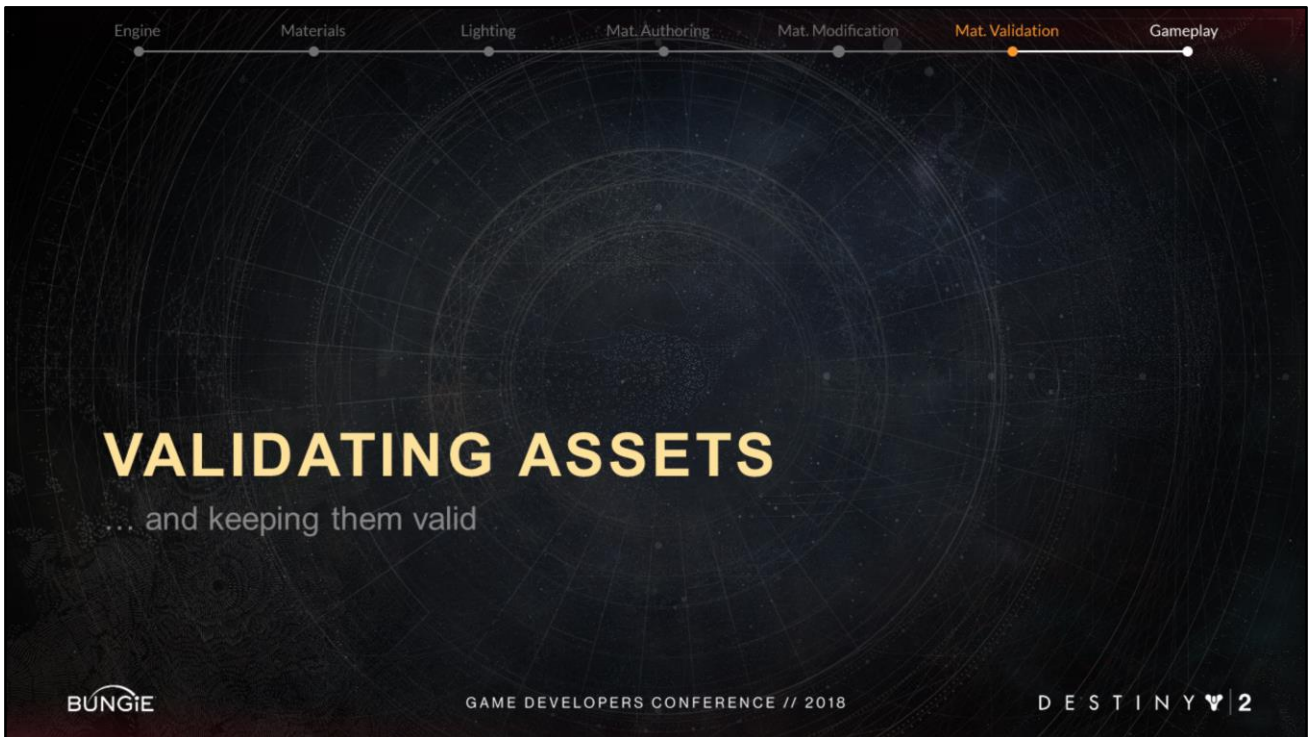
This turned out to be really successful. Artists no longer had to learn and memorize the idiosyncrasies of authoring a behaviorally convincing wet surface, with most content only opting to author the wetness amount parameter.



It was such successful example of safely simplifying complex behavior, that we built much of the game's opening mission with it!

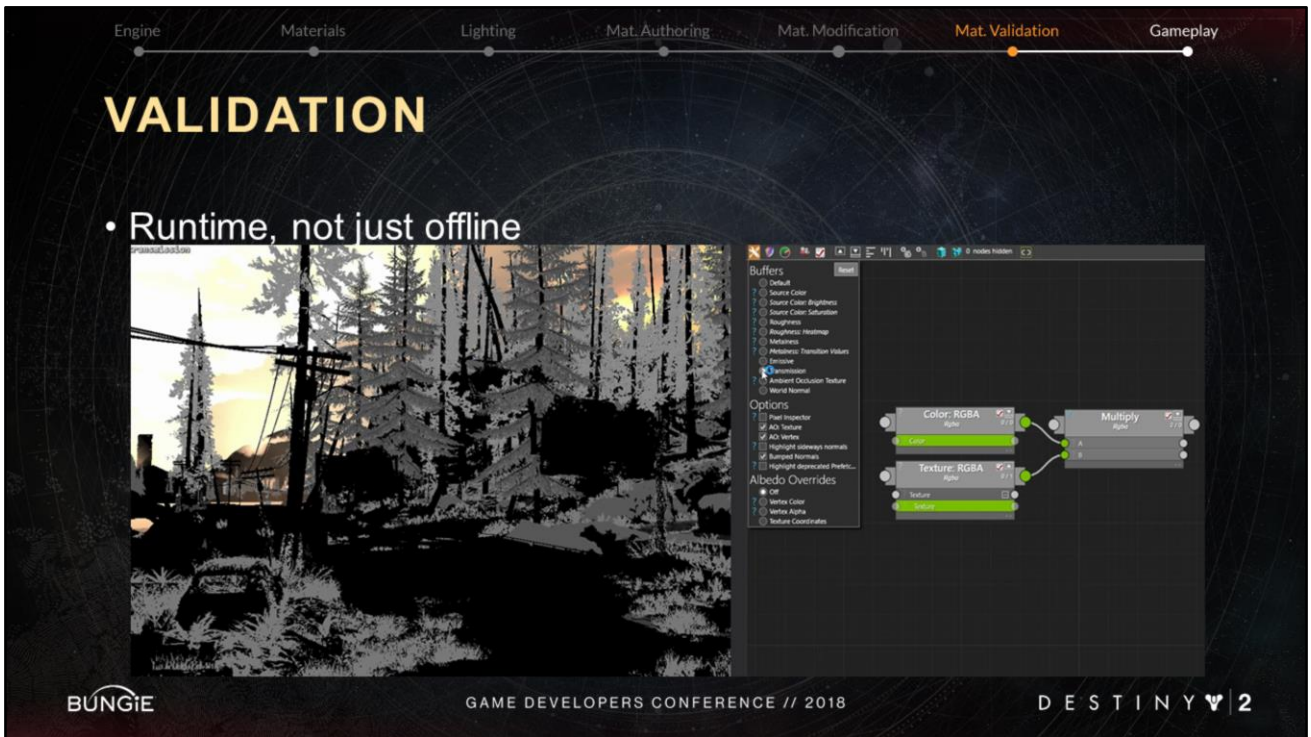
So at this stage we've gone from simple texture inputs, sometimes fed through complex and unrestricted shader node graphs, and postprocessed and modified with deferred decals and systems like wetness. But so many layers of transformation and so much work goes into a single pixel

how do we make sure each pixel output is still safe to a PBR ecosystem?



Validation is our answer to this.

This is one of the foundational elements that allows us some form of standardization and predictability of both the artist inputs we must be able to develop and extend graphics systems to, as well as the quality and behavior of our final rendered frame.



The lion's share of our validation is done at runtime. Runtime is really important as our assets are a mixture of texture AND gameplay-driven visuals. Ingame debug modes can be triggered at anytime from within the shader node graph itself. The goal here was to provide validation at the layer of content creation that the artist is likely already working.

In this video you can see the different options and validation modes made available to artists, with rich ToolTips continuing to be prevalent. Clicking on any of these will send a networked message to a running instance of the game and change things accordingly, this made validation less inconvenient for artists to follow through with.

VALIDATION: COLOR BRIGHTNESS

- Ensure colors aren't too dark/bright for their material

```
float blended_ref_color_min = lerp(0.029, 0.46, metalness);
float blended_ref_color_max = lerp(0.9, 1.0, metalness);
float source_max = max(max(source_color.r, source_color.g), source_color.b);
if (source_max < blended_ref_color_min) // Too dark
{
    ...
}
else if (source_max > blended_ref_color_max) // Too bright
```

The first of those validation modes was for Color Brightness, based on PBR-legal ranges corresponding to metalness.

This is fairly business as usual. Treat the metalness input as the claim to what the material-type is, and adjust valid ranges accordingly. Then, highlight using a colour that is unique to how the validation may fail.

Let's look at an example.



This is a final render pass from our game, and if I enable color validation

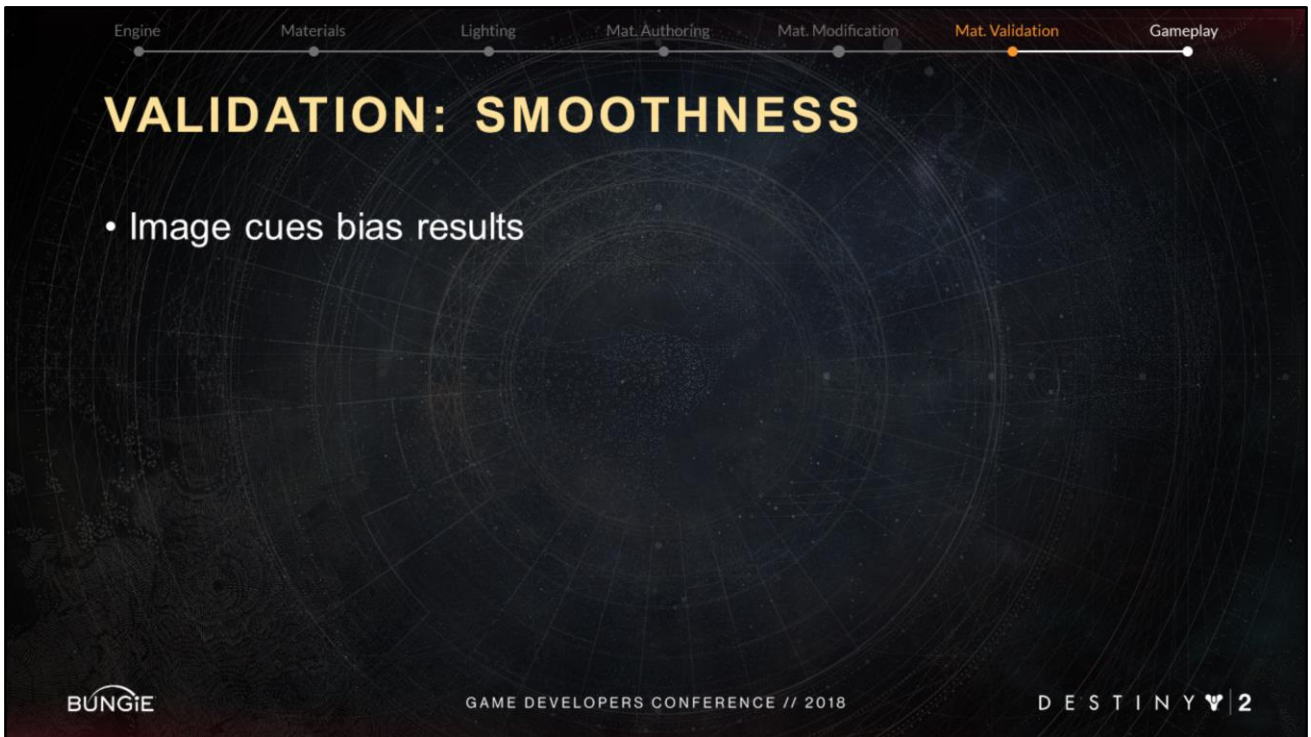


We can see offending areas being highlighted by color, with artist-reminders found on the upper left.

This helped to avoid sending them to an external resources for clarification of what each colour means.

We chose to use simple, learnable colour schemes, and *binary* highlights. Binary is a key word here, as it has the utility of simplifying interpretation that can vary from artist-to-artist, and day-to-day, on how 'wrong' something is OK to be.

When you're a Technical Artist at someone's desk asking them to change their artwork, using a neutral party like this can help to reduce some of the cultural friction or pushback you may encounter.

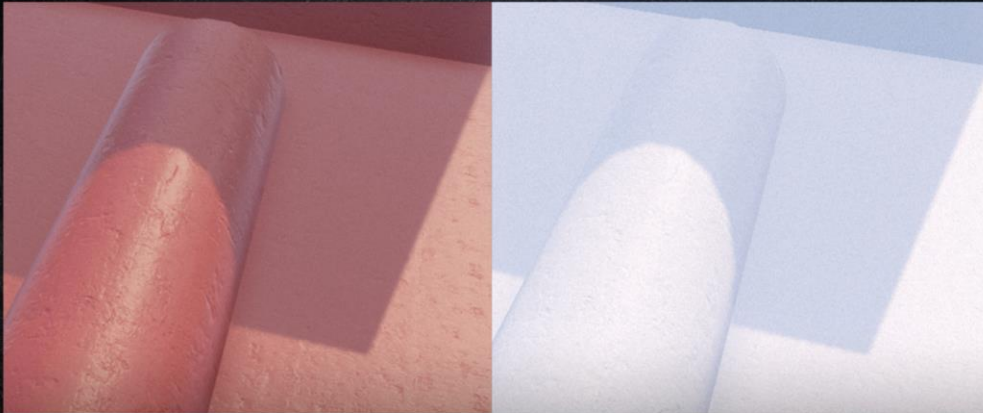


For smoothness, things are a bit different. A few distinct factors make this challenging for an artist that wants to make the right choice.

A person's understanding of surface properties is easily biased with image cues.

VALIDATION: SMOOTHNESS

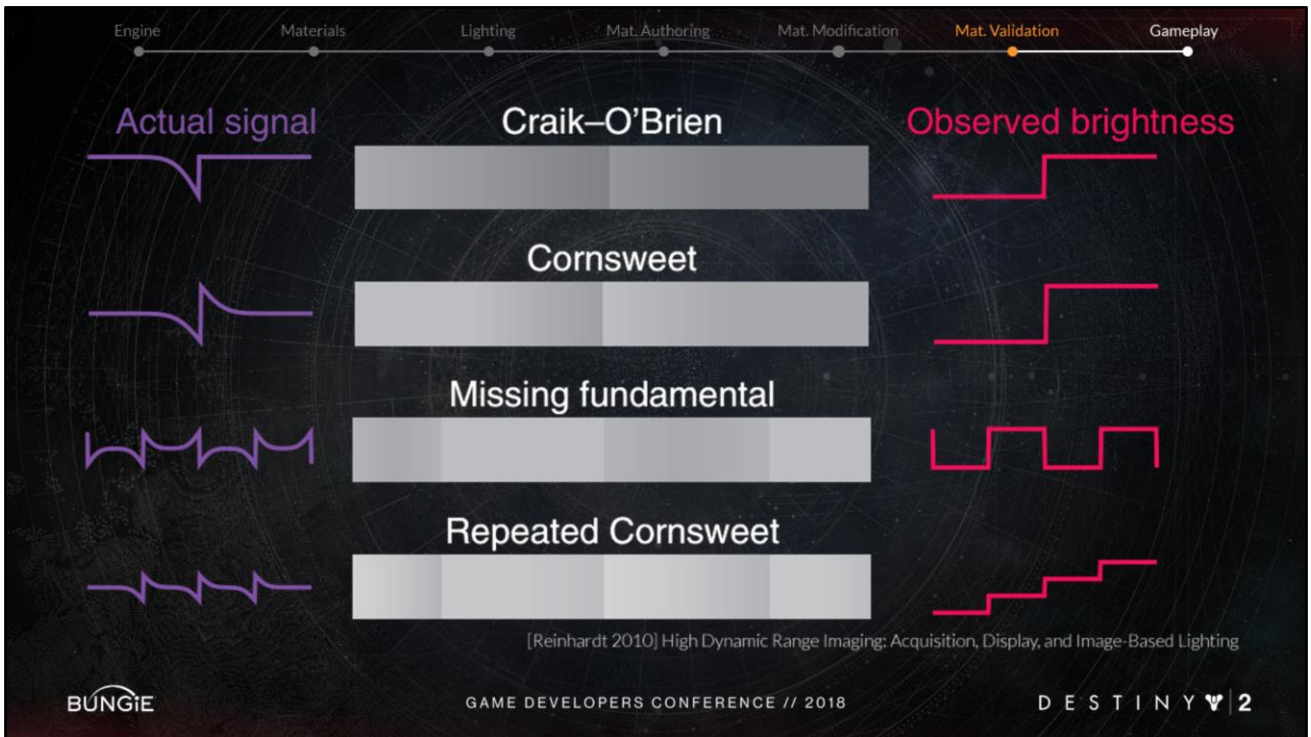
- Image cues bias results



In this example, these two cylinders use identical normal and smoothness inputs, the only difference is a red vs white albedo.

Despite this, the white version appears significantly rougher.

In the jewelry industry you would describe the white version as having very little luster. This is the apparent contrast between the reflected colour and the underlying surface colour working to push and pull mental models of what a surface is. Though the visual cues like the specular highlight are still present on the right example, there's very little contrast to appreciate it.



And this is something called the Cornsweet illusion. It starts with a solid fill grey colour. On the left you can see only small gradients being added, but on the right you can see the observed color deviates drastically from the true intensity.

Identifying absolute values that aren't neighboring can be nearly impossible.



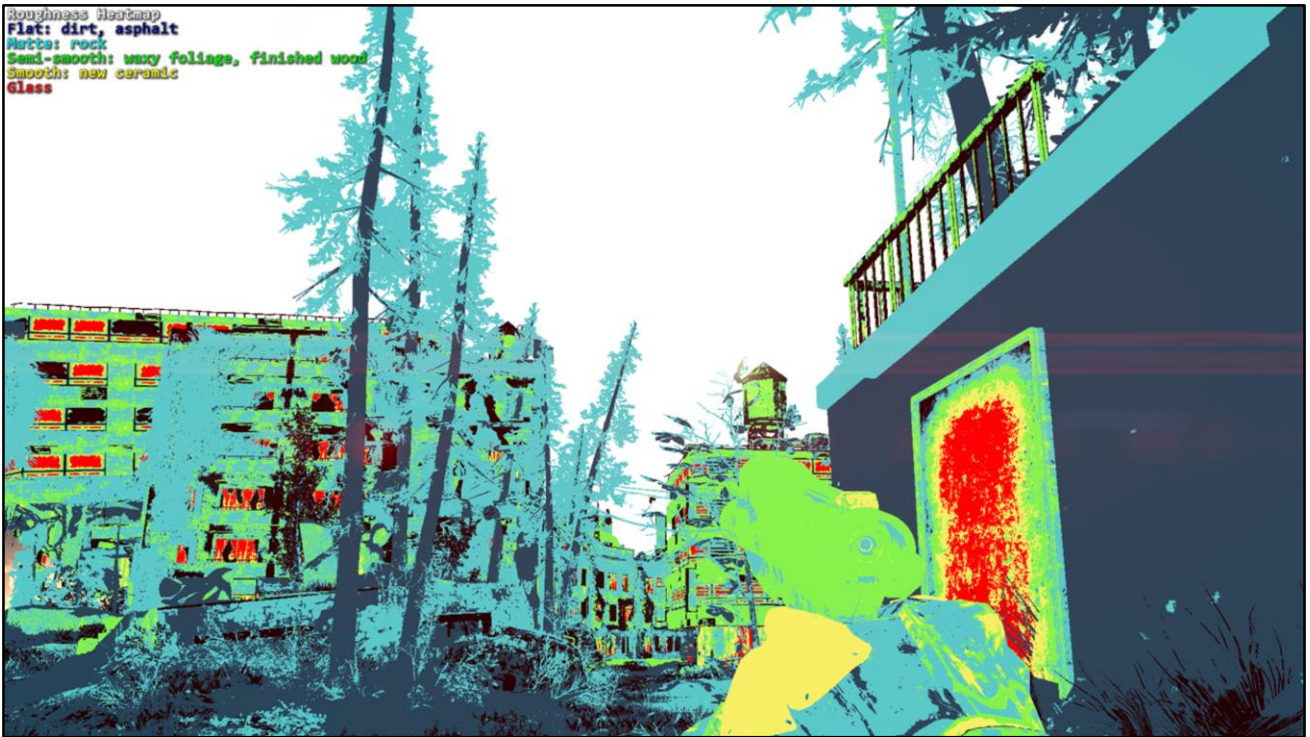
Yet it isn't uncommon to still ask artists to use *this* type of unintuitive, data debugging mode to confirm whether or not their art is authored correctly. This is not guiding them towards success.

VALIDATION: SMOOTHNESS HEATMAP

- Encourage 'families' of real world values
 - Based on paint manufacturer gloss terms
 - Flat, Matte, Semi-smooth, Smooth, Glass
- +Bonus: Terms are learnable by production

Our approach attempts to take these biases out of the equation. Values are divided into five discreet categories and abstracted into a heat map, using terms inspired by the paint manufacturing industry.

We found it helpful to actively establish clear language for abstractions like this; providing a more precise shared language between artists and directors.



This is an example of it running in our game. Again we embed artist reminders of relatable materials found into the upper left.

We did have a few more emergent benefits of these validation modes. Because they're so heavy handed in their visuals, we were able to do Technical Art audits of large areas and look for obvious issues. Even when you're looking through a space quickly, seeing red in a primarily natural art palette easily stands out to investigate.

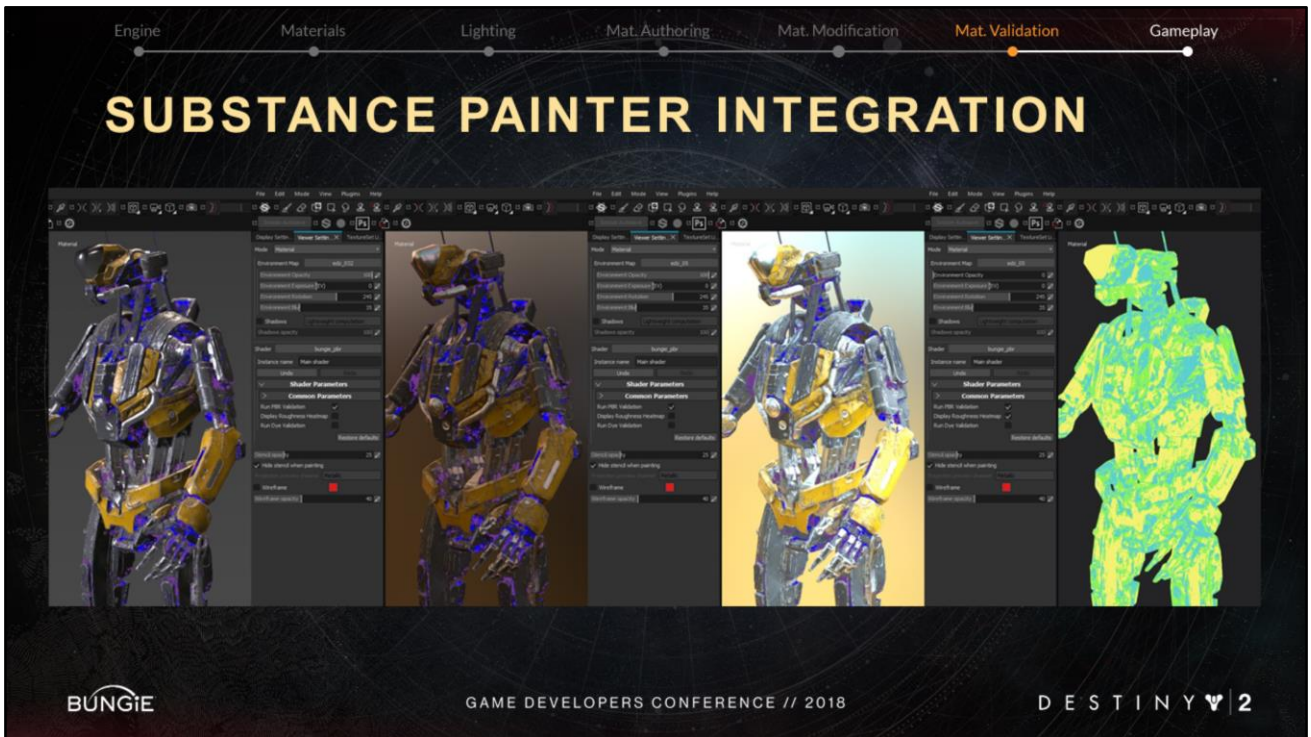
(in this image example, the red is actually glass windows)



We also run nightly captures of specific content in validation modes to catch emergent issues. This was actually implemented by our resourceful test team, using automation and debug hooks we had already developed for artists.

In this example you can see an image we generate that showcases a shader item the player can use to customize their Gear, and ensure that assets are within guidelines.

This helped to provide enough **guidance** to empower test resources to report Art bugs

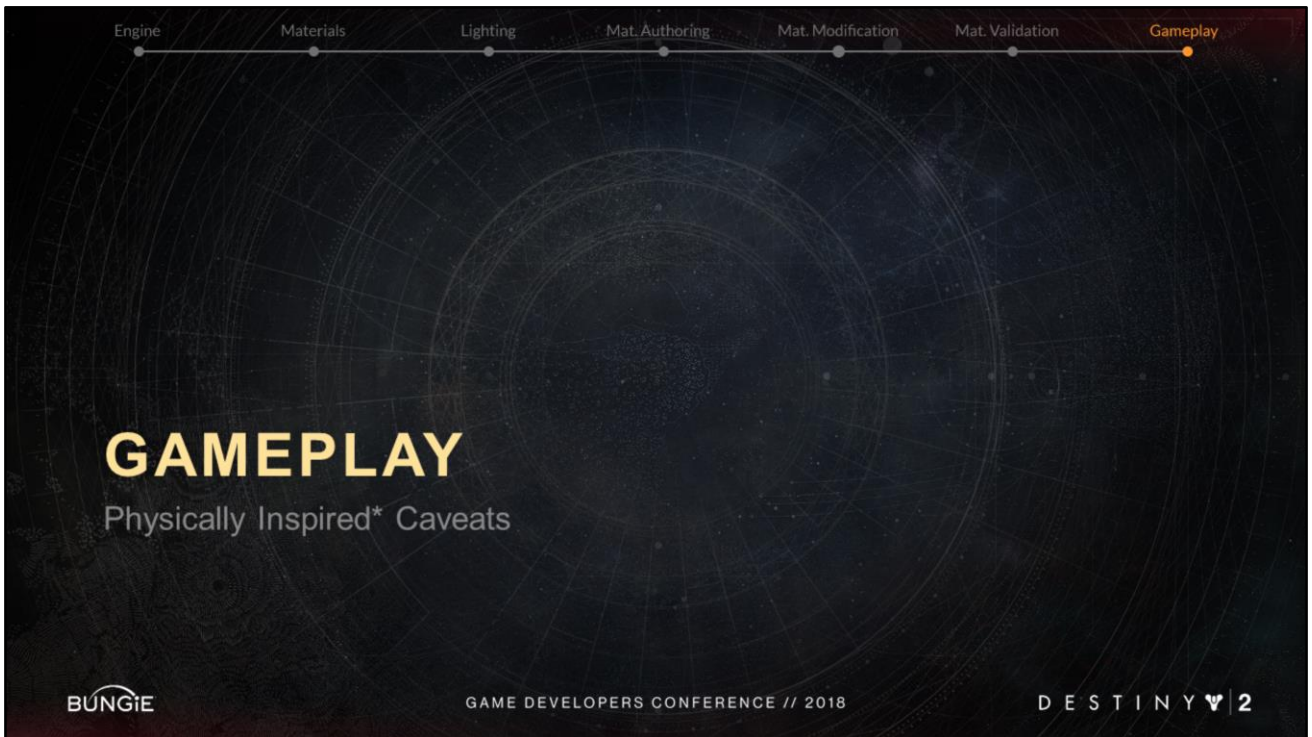


In addition to runtime validation, we also built this into Substance.

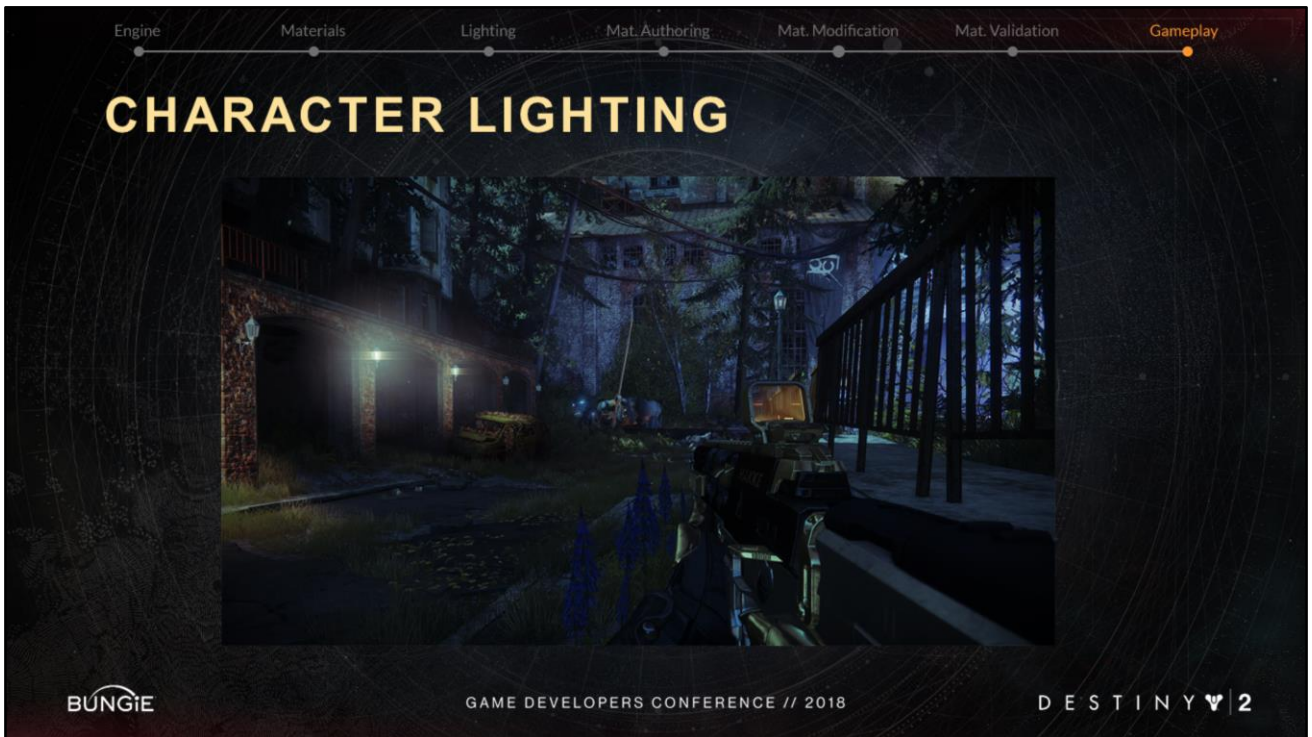
In this example we have a character loaded up in Substance Painter, with various in-game IBL captures being used to light the scene and different validation modes being enabled.

If you haven't already done this in your development, please take another look. It takes very little work to move your HLSL to a place where your artists are likely creating their content already.

And that's closes our validation section. Ultimately we found that we can provide guidance that doesn't require memorizing magic numbers and parameter co-dependencies, it can be simple colors with plain English descriptions. And the closer you can move these checks to the decision making itself, the more successful you'll see the adoption of them.



Now that we have physical materials, we're nearly done. The last thing we have to do is support our gameplay needs that begin to stress the usage of the words *Physically Inspired*. Needs that require systemic solutions that can peacefully co-exist with the rest of our PBR ecosystem.



The first wasn't all together surprising once it surfaced.

PBR is often looked at as a way to get a coherent frame where things seamlessly blend with realistic lighting.

However, this can create your own worst enemy if you value quick identification of enemies onscreen.

If you hadn't realized already, there's actually an enemy in the direct center of the screen, but sometimes even lens flares can't solve this problem



So we knew that we would have to start cheating PBR here, if we're going to bias PBR, we might as well standardize the bias too!

Creatively named, "Character Lighting"

It has minimal artist-facing parameters by design, and it effectively causes all characters to automatically become emissive when seen at a distance.



And now we can see our enemy-

Since our Gbuffer only stores a single Emissive scalar that multiplies the albedo, it caused characters with a brighter luminance to glow faster than others, so we do a slight inverse-luminance compensation from the source, then further attenuate over distance and an object space gradient, to avoid having glowing feet making it look like our characters were floating.

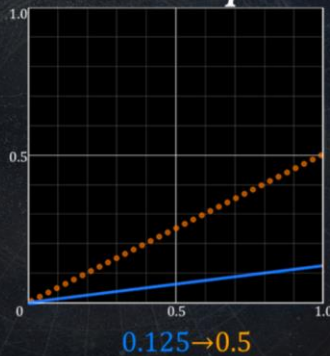
TRANSPARENT EXPOSURE

- “Hollywood” explosions

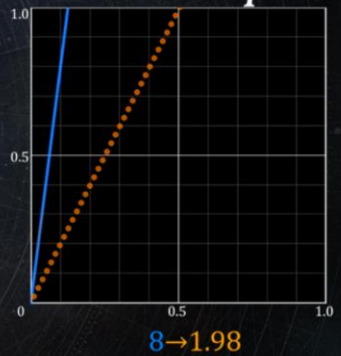
$$\text{Exposure} = 2 \text{ stops}$$

$$\text{Transparent Exposure} = 2 \text{ stops} \cdot \text{slope}$$

-3 stops

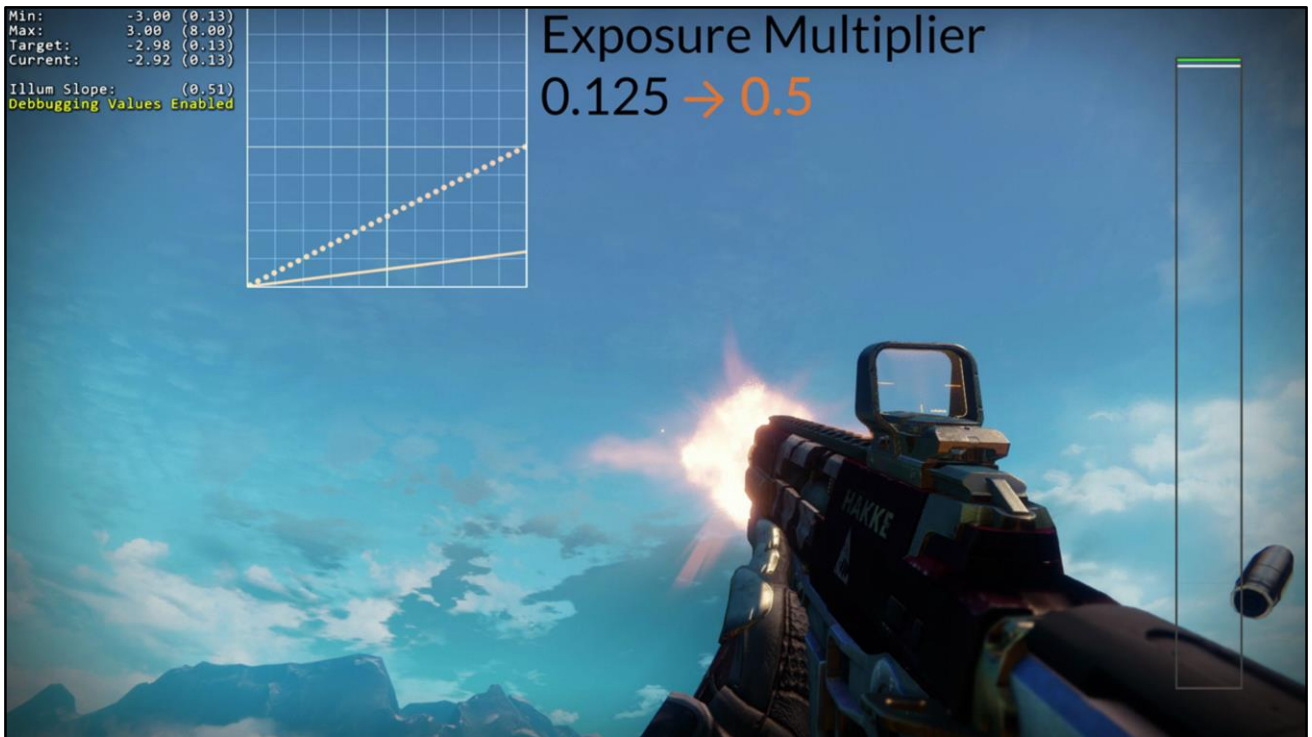


+3 stops



The next was a desire to make emissive content feel more “Hollywood”. What that translated to was explosions and muzzle flashes more pronounced in bright conditions, but not blinding in dark conditions. Initially artists pre-translated this as a request for per-shader exposure control. After talking it through, we opted for a more systemic approach.

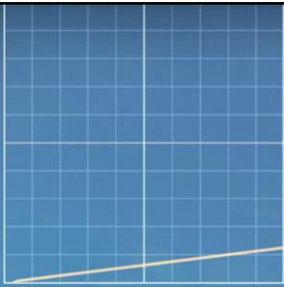
We apply a biased-slope to the exposure calculation for emissive transparents I know I just used the dreaded word bias. There is likely future opportunity here that involves revisiting our quadratic bloom curve, tonemapping, and content intensity standardization. For Destiny 2 we opted to scale the target exposure stop by 0.33, allowing us to undershoot or overshoot exposure depending on conditions.



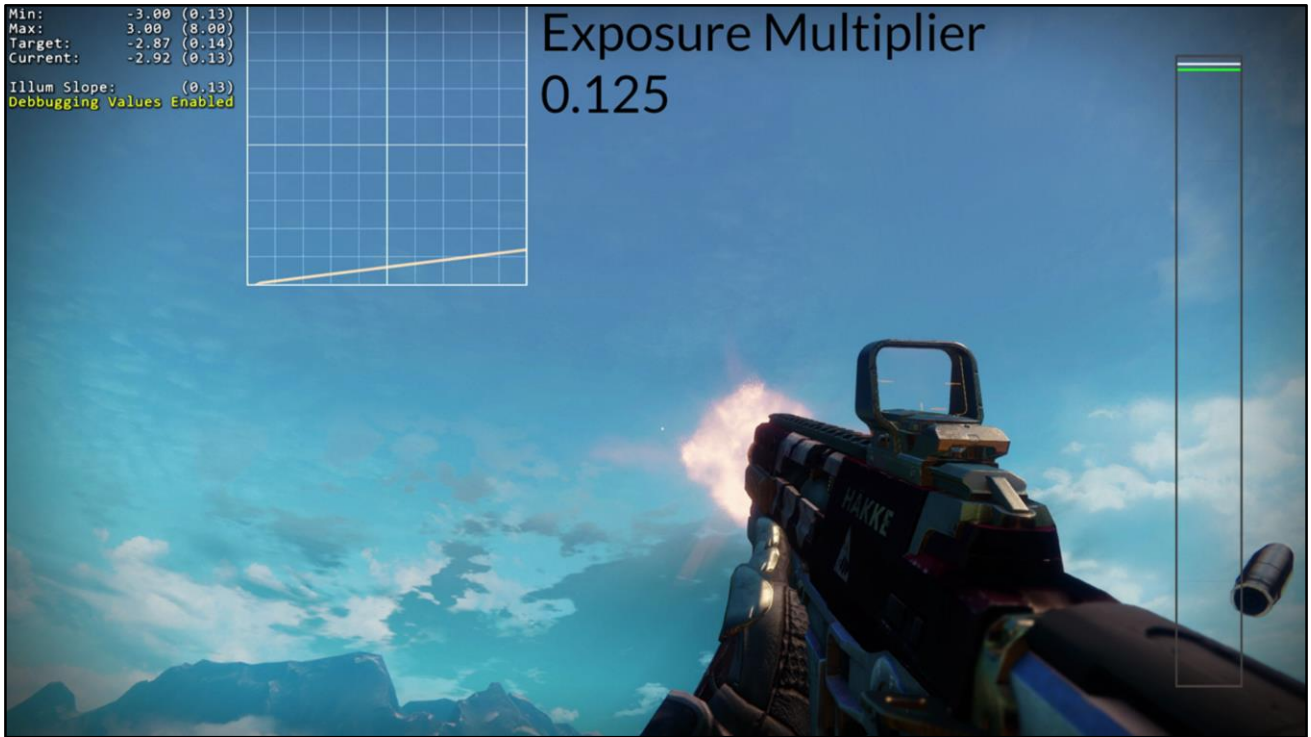
In bright contexts where your target exposure stop is -3, our exposure scaler increases from 0.125 to 0.5.

Making muzzle flashes more noticeable

Min: -3.00 (0.13)
Max: 3.00 (8.00)
Target: -2.87 (0.14)
Current: -2.92 (0.13)
Illum Slope: (0.13)
Debugging Values Enabled



Exposure Multiplier
0.125





In darker contexts where your target exposure stop is positive 3, our exposure scaler decreases from 8.0 to 1.98

Making muzzle flashes retain their detail, and allowing the player to see what they're shooting without blinding them.

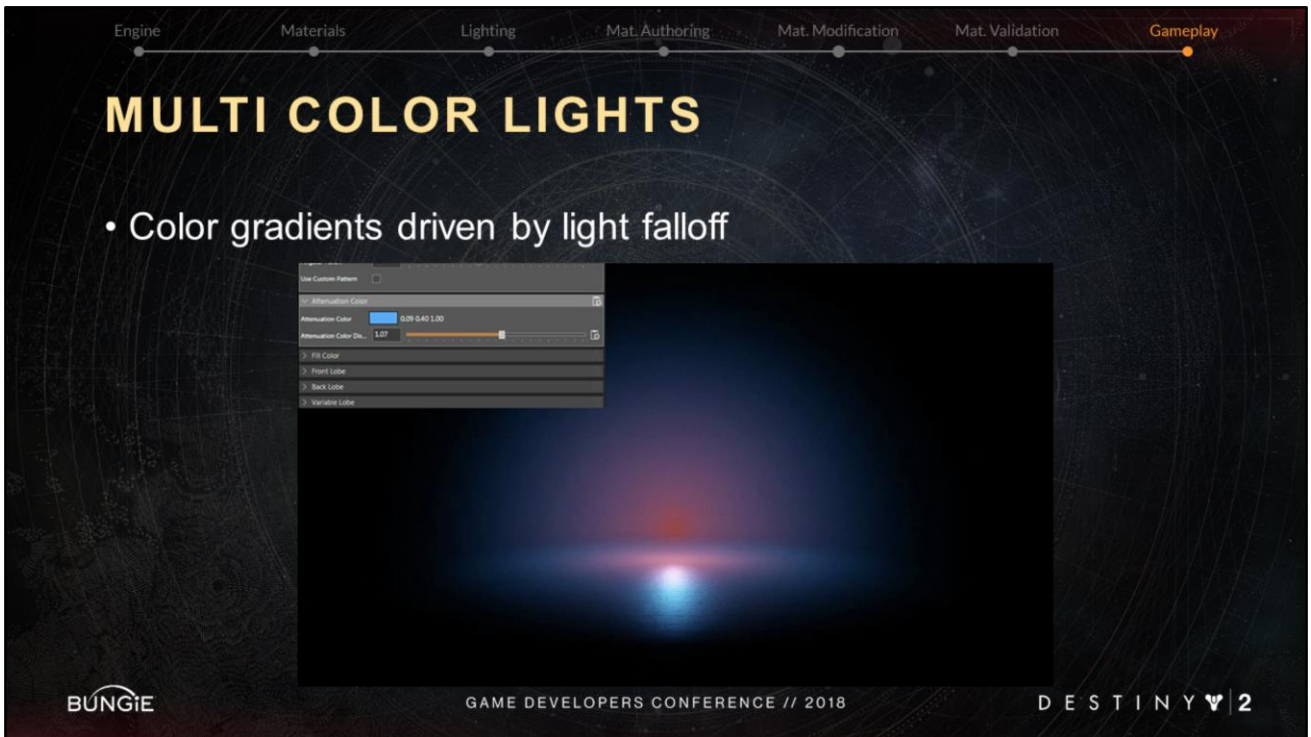
-

Technically this did expose us... .. to the risk of **auto**-exposure oscillations due to the feedback loop of the bias, but in practice the affected content doesn't spend much time on screen so it didn't significantly affect us.

Min: -3.00 (0.13)
Max: 3.00 (8.00)
Target: 2.20 (4.61)
Current: 2.92 (7.56)
Illum Slope: (7.56)
Debugging Values Enabled

Exposure Multiplier 8.0



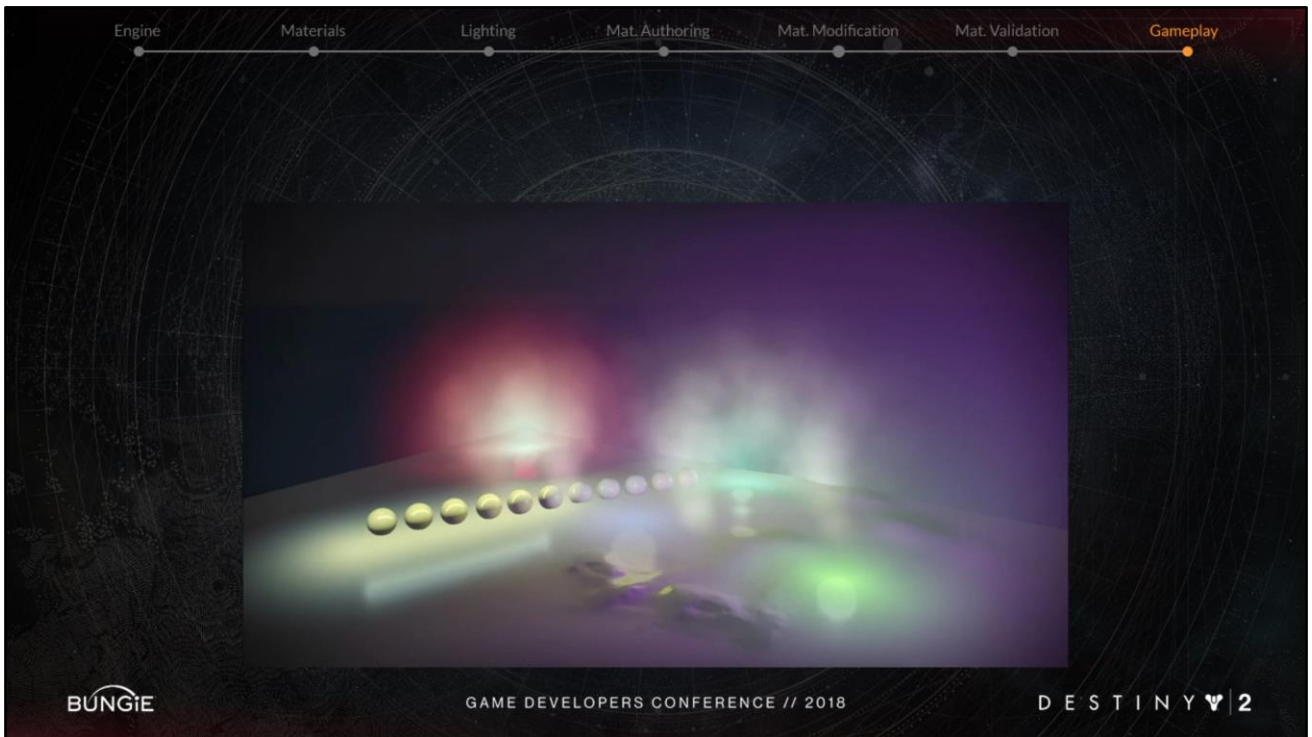


The next was a desire to provide greater colour variation to lights for gameplay purposes, but it clearly wasn't practical to double our light count.

So we allow artists to provide a secondary radiance colour that we lerp to over the squared falloff.

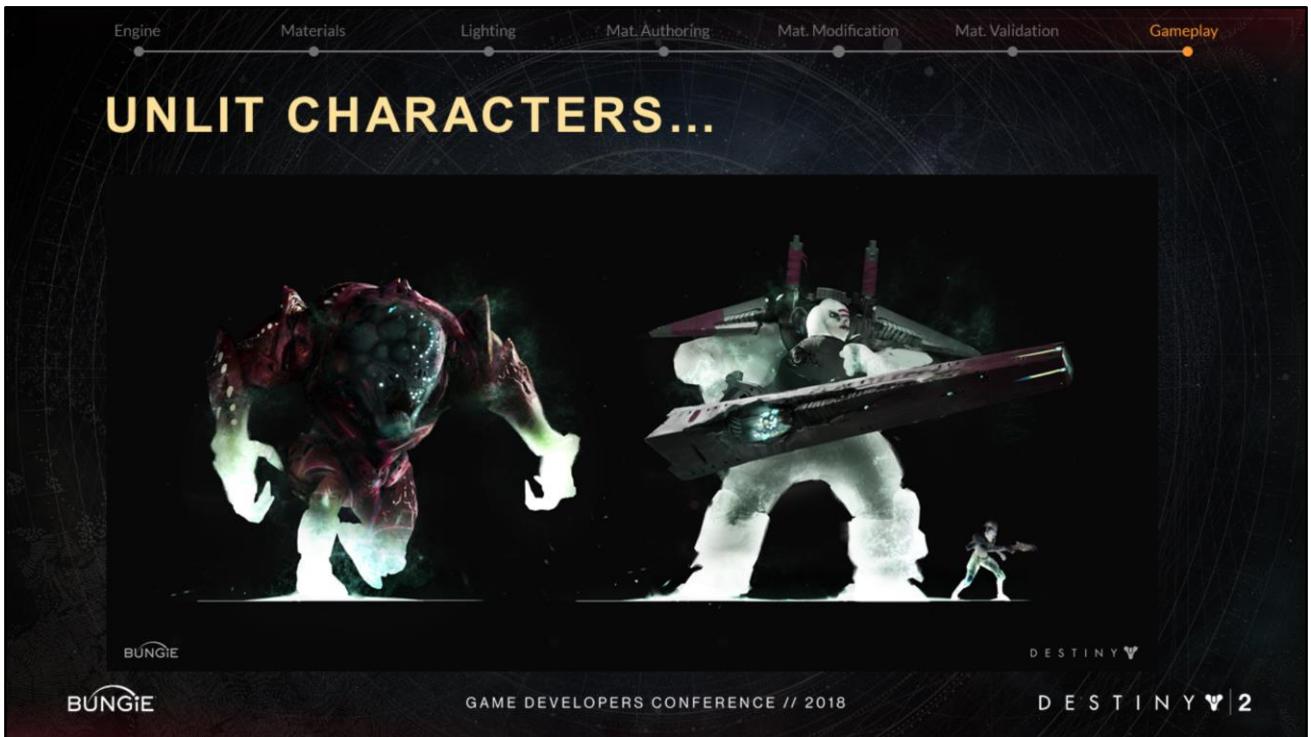
To try to rein this closer to the plausible side of things, the outer colour is limited to the luminance of the inner colour.

In this example you can see the colour represented in the volumetrics as well.



When combined with several volumetric lights and particles rendering to volumetric buffers, you can fairly quickly create compelling imagery, as demonstrated by my test level that is sporting an ambitiously black sky and grey floor. Like any other leaf feature for special situations, this ended up being used in nearly every lit scene and dynamic light... ..

But that's just a sign of a good feature, right? 😊



So for the last thing we'll cover today, I wanted to briefly mention the last part of wrapping up our PBR implementation

We have a combatant race in Destiny called The Taken, the art direction is that they look **Other Worldly**. As it turns out, **Other Worldly** isn't a parameter that you can find in the original Disney Brent Burley PBR papers.

Translated, **Other Worldly** just means, "turn off all lighting and add emissive content"... ..

UNLIT CHARACTERS...

```
float lighting_crush = (iridescent_id >= 0.995) ? 0.0 : 1.0;  
diffuse_irradiance   *= lighting_crush;  
specular_irradiance *= lighting_crush;
```



BUNGIE

DESTINY

BUNGIE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

So, we ended up reclaiming one of our Iridescent IDs to dynamically branch in our shading pass. When this is enabled within a node graph, the surface opts out of all incoming diffuse and specular irradiance.

Current hardware handles coherent branches like this very well, and it's encouraging towards future shading model divergences with the other remaining utility bits.

Engine Materials Lighting Mat. Authoring Mat. Modification Mat. Validation Gameplay

UNLIT CHARACTERS...

```
float lighting_crush = (iridescent_id >= 0.995) ? 0.0 : 1.0;  
diffuse_irradiance   *= lighting_crush;  
specular_irradiance *= lighting_crush;
```

- ✓ Simple Inputs
- ✓ Safe Material Postprocess
- ✓ Output Validation
- ✓ Space Magic

BUNGIE DESTINY
BUNGIE GAME DEVELOPERS CONFERENCE // 2018 DESTINY 2

And with our unlit characters, we close out a successful transition to PBR, while still maintaining our art diversity, and most importantly... our space magic.

TAKEAWAYS

- **Safety:** Usually perf-affordable
- **Simplicity:** More control != More better



[Bennett Foddy] QWOP

So to close things out, here are some takeaways we learned while developing for Destiny 2's unique constraints and direction.

Safety and simplicity are affordable. Current hardware can afford extra cycles needed to build simpler interfaces, protecting artists against divide by 0s and other unsafe problems that were previously their burden.

More control is rarely better due to added complexity. Exposing every parameter requested by art can provide short term benefits, but left unchecked can lead to a degradation of standards, artist confidence, and can deprive everyone solutions to the real problems being faced.

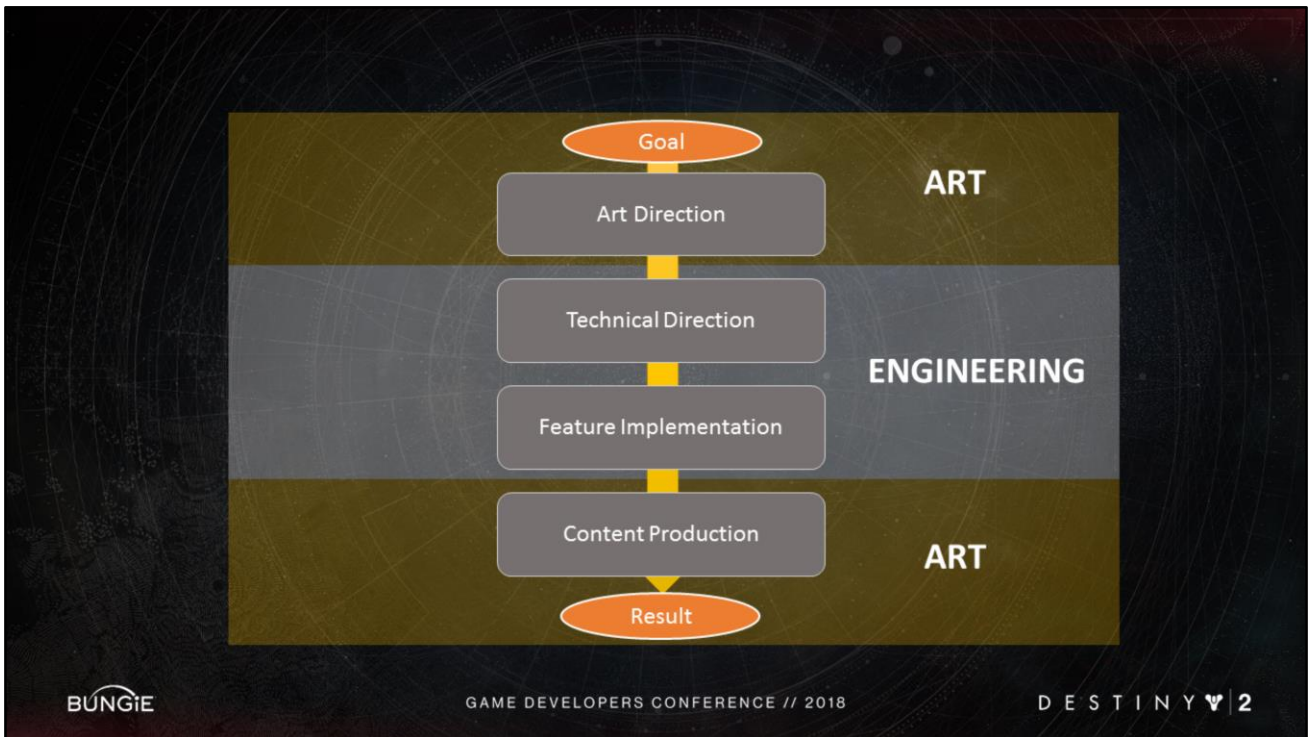
TAKEAWAYS

- **Safety:** Usually perf-affordable
- **Simplicity:** More control != More better
- **Guidance:** Education is critical
- Write code close to content

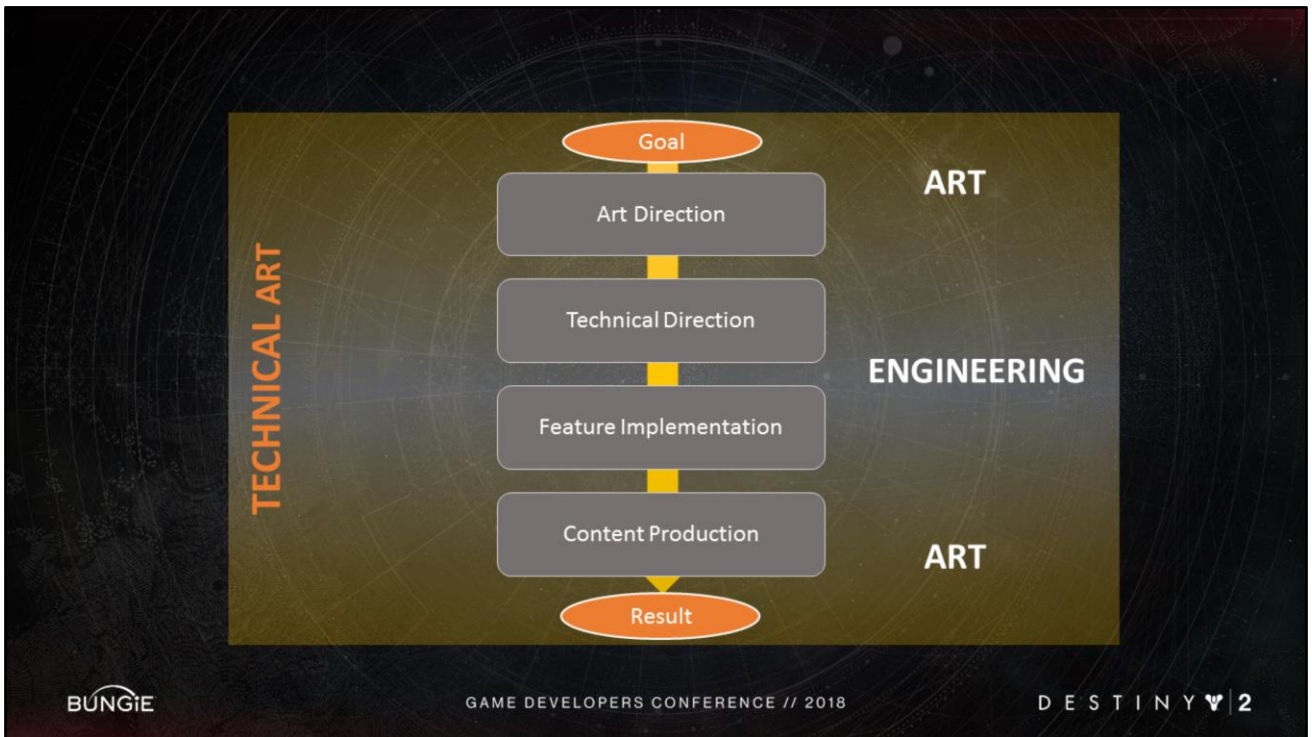
Even implementing a standard system like PBR will require solutions that are unique to your game, and this is only half the work.

The second half of rolling things out is just as important. Have a plan for artist education and documentation. Hold theatre presentations, classroom sessions, identify people that can help evangelize and champion changes within your studio.

And lastly, consider developing code closer to your content.

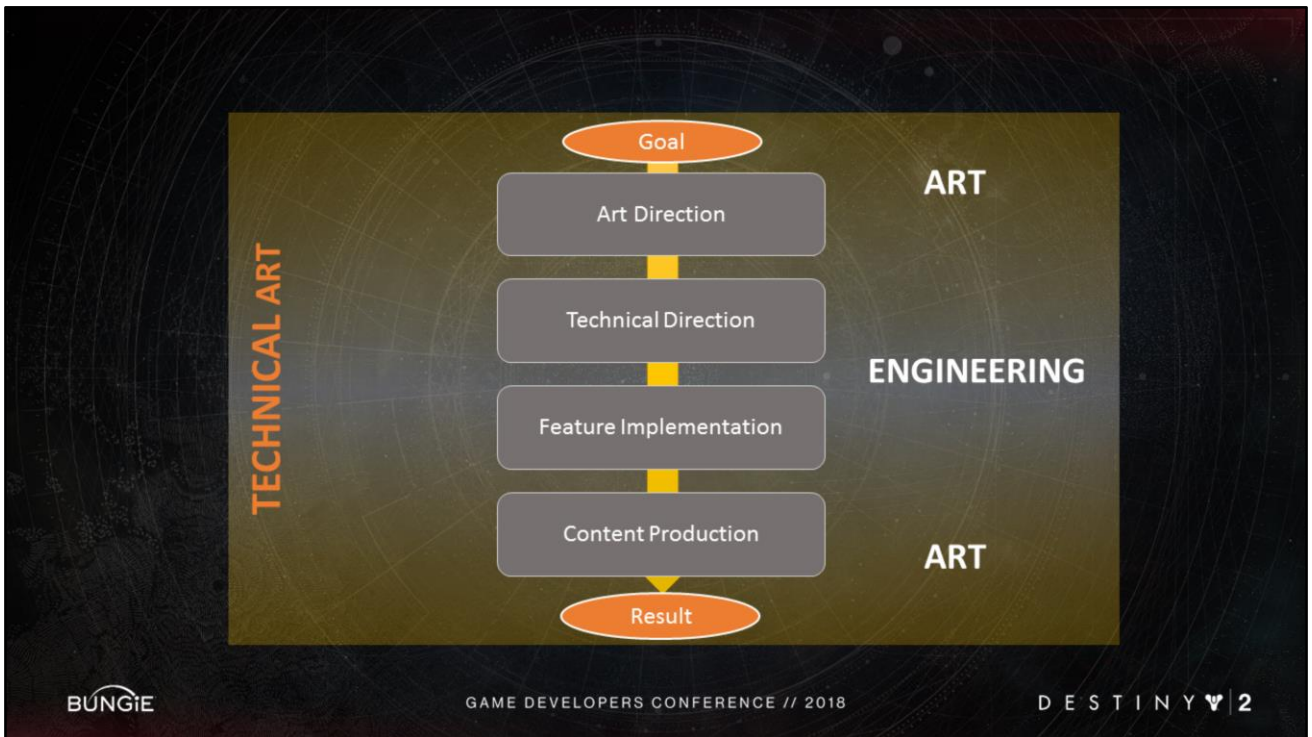


In the past, we would sometimes find the lines between Art and Engineering to be well defined, leading to sometimes cumbersome interfaces, and technology that wasn't as well aligned with our actual project's goals, due to the signal loss introduced between each stage.



But by blurring these lines during Destiny 2, we found that Technical Artists can write production code if you give them a safe environment like TFX does.

This blurred area in-between **is** Technical Art at Bungie, and I think that we as an industry as just now starting to realize what it means for Engineering to deliver a technology, that still requires molding and adapting to be well leveraged for your game's content, we improved this just by moving development even closer to production artwork.



... and this is now how we develop **all** of our graphics features at Bungie
With Alexis' help, I was able to pitch in to implement some of the features we talked about today, and I'm just a TA!

THANK YOU

- Bungie Graphics and Art teams
- Tom Burlington
- Chris Tchou
- Ryan Ellis
- Natalya Tatarchuk

BUNGIE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

And thank you of course to the Graphics and Art teams back at Bungie, and the people beyond Bungie that helped steer us along the way.

... and now I'll turn it over to questions

QUESTIONS?

Contact

aharoux@bungie.com

nhawbaker@bungie.com

BUNGIE

GAME DEVELOPERS CONFERENCE // 2018

DESTINY 2

W E ' R E H I R I N G



WWW.BUNGIE.NET/CAREERS
CAREERS@BUNGIE.COM

- We're hiring, including positions on our specific team!
- Fill out the evaluation survey for the talk
- If you want to talk more about graphics in Destiny 2, or feel passionate about developing a TA org, we'll be hanging out near one of the Wrap Up rooms

REFERENCES

- [Burley 12] SIGGRAPH 2012: Physically Based Shading at Disney
- [Tatarchuk 13] SIGGRAPH 2013: Mythic Science Fiction In Real-Time: Destiny Rendering Engine
- [Tatarchuk & Wang 14] SIGGRAPH 2014: Creating Content to Drive Destiny's Investment Game
- [Lagarde 14] SIGGRAPH 2014: Moving Frostbite to Physically Based Rendering
- [Lagarde 17] SIGGRAPH 2017: Physically-Based Material, where are we?
- [Reinhard 10] Reinhard, E. Ward, G. Pattanaik, S. Debevec, P. Heidrich, W. Myszkowski, K (2010) High Dynamic Range Imaging: Acquisition, Display and Image-Based Lighting
- [Tatarchuk & Tchou 17] GDC 2017: Destiny Shader Pipeline
- [Bennett Foddy 08] QWOP