
# GEARS OF WAR 4: CREATING A LAYERED MATERIAL SYSTEM FOR 60FPS

COLIN PENTY, THE COALITION
IAN WONG, THE COALITION

# INTRO

- Colin Penty
  - Technical Art Director. Has been working @ The Coalition for 6 years.
  - Previously a TAD @ EA for 7 years working on Need for Speed, Skate, FIFA.
- Ian Wong
  - Senior Rendering Engineer. Has been working @ The Coalition for 6 years.
  - Previously at Propaganda for 5 years working on Tron, Turok.

# GEARS OF WAR 4 OVERVIEW

- The Coalition
  - Acquired Gears IP from Epic January 2014
- Gears of War 4 Shipped Holiday 2016 on Xbox One and PC
- Ran in 1080p 30fps in campaign and 1080p 60fps in multiplayer
- Shipped using Unreal Engine 4.11 with custom modifications
- 84% Metacritic after 99 reviews on Xbox One
- “A technological masterpiece for both Xbox One and PC” – Digital Foundry


Hey guys, so before we started I wanted to give a quick overview of Gears of War 4 and The Coalition.
The Coalition acquired the Gears of War intellectual property in January 2014 from Epic Games.
We immediately began work on Gears of War 4, and shipped the game this past holiday on Xbox One and PC.
We were able to ship at 1080p, 30frames per second in campaign and 1080p, 60 frames per second in multiplayer.
We shipped using Unreal Engine 4 4.11 with many custom modifications and Xbox One optimizations.
After 99 reviews we are at 84% metacritic on Xbox One.  Game has reviewed well.
Digital Foundry wrote a very flattering piece on Gears 4 stating it was “A technological masterpiece for both Xbox One and PC” which made us tech guys very happy.

# COMPONENTS AND TALKING POINTS

- *MMS = Material Masking System*
- Material Container UI and workflow in UE4
- Material Generator in UE4
- Material Cooking system in UE4 (Smelter)
- Numerous secondary systems.
- Challenges, Wins, and Learnings

Components and talking points:

First, we call the system internally the “MMS” system which stands for Material Masking System.
It consists of a few elements that we will speak about:
-Our Material Layer stack which we call the Material Container. This was a custom UI we built in UE4.
-The material generation system which catenates a material together based on content creator inputs.
-The material optimization system which we call the Smelter which cooks our materials down.
-Numerous supporting systems that Ian will be speaking about.
-Finally we will be speaking about challenges, wins, and learnings with MMS.

# THE WHY

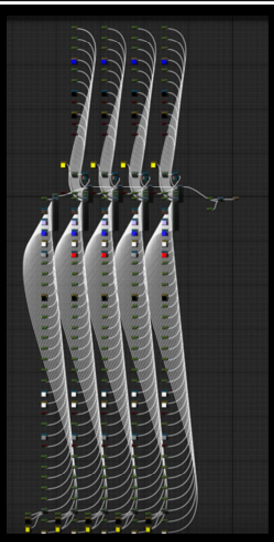
1. Inspired by Ready at Dawn's Material work from The Order: 1886 [Neubelt and Pettineo 2013]
2. Keep content creation in-engine as much as possible
3. Up front enforcing of PBR values and texture authoring pipeline
4. Real-time feedback of material parameter tuning
5. Have to hit 1080p, and 60fps in MP. Need efficient content.

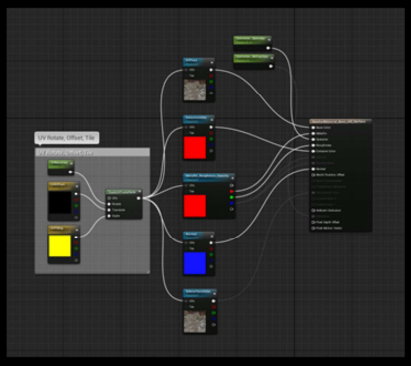
So why did we build an integrated material layering system? Doesn't Unreal Engine have a great material system out of the box?
When I get back into my headspace in 2014 when work began on our material layering system – a few things were top of mind for me.

1. I had just seen Ready at Dawn's material presentation at SIGGRAPH that looked amazing and elegant for The Order: 1886. Loved their approach.
2. I was noticing a trend that more and more content creation was moving into engine that once wasn't. I was wondering if we could take the next natural step on material creation.
3. I had been recently working on Gears of War: Ultimate Edition, and noticing some real asset consistency issues with our content. Artists were sometimes putting AO and other detail into diffuse textures, and PBR calibration was very inconsistent. At scale we need more than just best practices – we need new tools and workflows.
4. Given the added complexity of our PBR materials, artists were having a harder time "guessing" what an asset will look like in Unreal. Adding real-time tuning in-engine to all of the materials that make up an asset removed all the guessing games.
5. Having the goal of hitting 1080p 60fps on Xbox One was pretty terrifying.

I wanted to do everything I could to mitigate the content performance risk.
This is where the idea of cooking our materials down came from.

# IN AND OUT





So, what goes in and what comes out of the smelter.
On the left is the “unoptimized” material the MMS system generates as the artist is building their material in the Container UI.  On the right is the material after it has been smelted down.  This is what is used in the run-time.
You can see the massive difference in complexity.

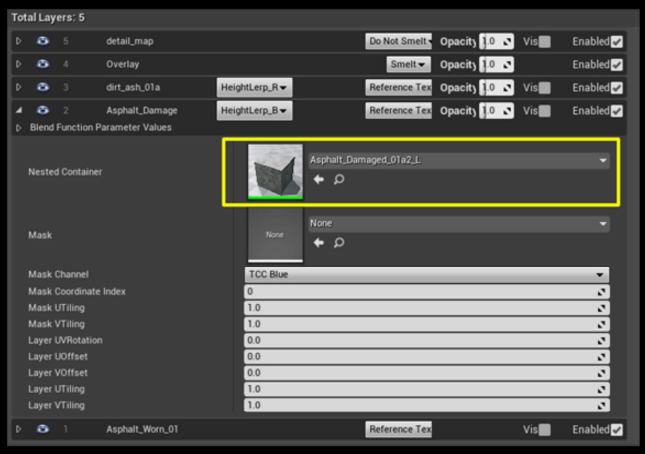
# HOW DO WE USE IT

- Artists generate material library
- Used on Environments, Characters, Weapons/Vehicles
- Tech Artists generate material functions and choose parameters to expose.
- Artists use MMS UI to assemble a material using Material Library, Material Blend Functions, and Masks.
- Processes (Smelts) material on save in seconds

Here is our common content production flow.
The artists generate a material swatch library using Material Containers. Currently we have 164 material swatches that make up all of our environment materials for example.
We can apply these swatch libraries to environment assets, characters, and weapons.
The tech artists populate the database with various material functions that can be used on any given layer in a container.
The artists use the container UI workflow to assemble a material by importing in masks, assigning the appropriate material function to a layer, then assigning the appropriate swatch and overrides.
Once the artist hits save on the material container the material is smelted and the output textures are generated.

# MATERIAL NESTING

- Artists can create a common material stack
- Then nest this into another material stack
- Live tuning of nested or non-nested materials
- Immediate propagation through database
- Allowed us to create global material library

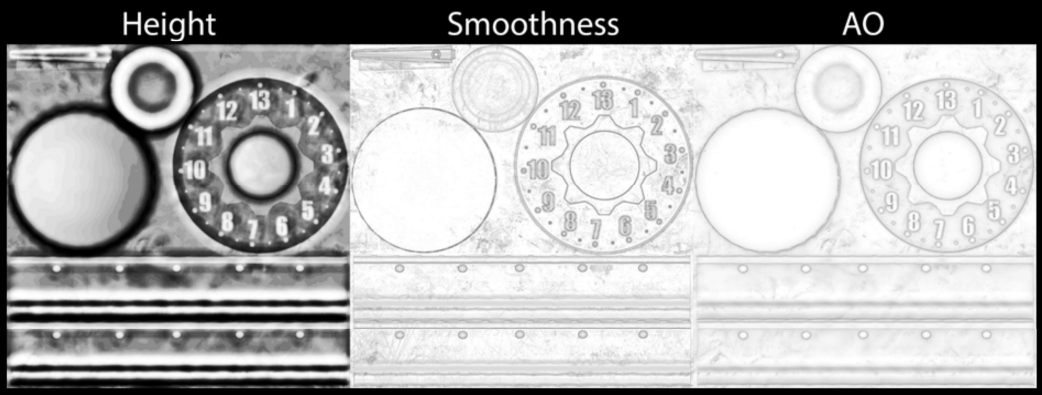



We also added the ability to have a material container nested inside another material container which really opened up interesting workflow possibilities. You can see in the image here the Layer called “Asphalt_Damage” has a nested container assigned (in the yellow box). All of our material swatches are used in this way but we can also nest more complex materials if desired. We support live tuning of the nested container parameters and the primary container parameters as well.

What’s nice is that since the whole system is live referenced all the time, the moment you modify a nested container, those changes propagate through the material database immediately. We rely on automated smelting scripts to catch any unsmelted materials and smelt them down as soon as a change as gone in.

# SHADING TEXTURE

- Auto generated texture from normal map. Done in UE4.
- Contains Height/Smoothness/AO
- Does not make it to the game directly. MMS system processes it.

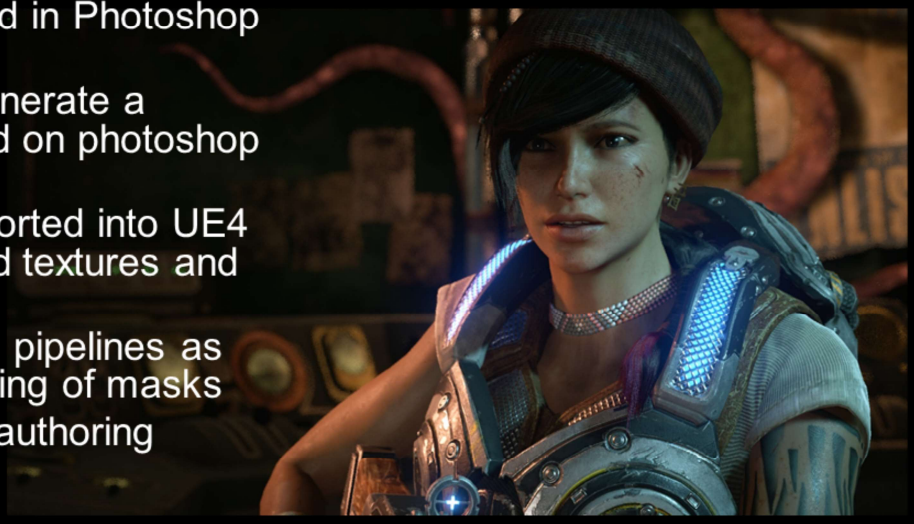



All of our primary material functions have an input for a shading texture. Since we know our materials will be cooked out by the smelter we can have texture inputs that are “helper” textures that are a great way to enforce quality and consistency.
Once an artist has a normal map in the unreal editor they can right click on it and the system will auto generate a shading texture for them.
The texture consists of a height map, smoothness map (yes we’ve inverted roughness for ease of use), and Ambient Occlusion. The auto generated smoothness represents a macro facet smoothness – we rely on the swatches to define microfacet smoothness – the two get combined on smelt.
Getting the AO quality and Height Map quality especially to our liking took a lot of iteration with the art team. The height maps are primarily used for vertex colour painting.
Little known fact – you can see this clock has 13 numbers on it instead of 12. The planet Sera in the Gears of War universe has 26 hours in a day!

# CONTENT CREATION WORKFLOW



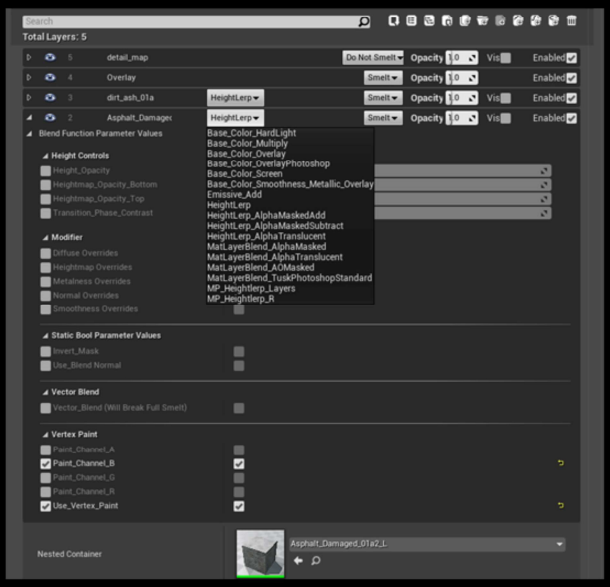
- Masks are generated in Photoshop or Substance.
- System can auto-generate a material stack based on photoshop layers
- Masks are auto-imported into UE4 as 4 channel packed textures and assigned
- We have Substance pipelines as well for quick importing of masks
- Majority of material authoring happens in Unreal



The artists will need to feed the system with masks – these are created in Photoshop or in Substance for the most part and imported into Unreal.
We have batch scripts that will auto-generate arrays of masks and generate material containers in Unreal from those masks.
As a result of this workflow the majority of the material authoring is done in Unreal.

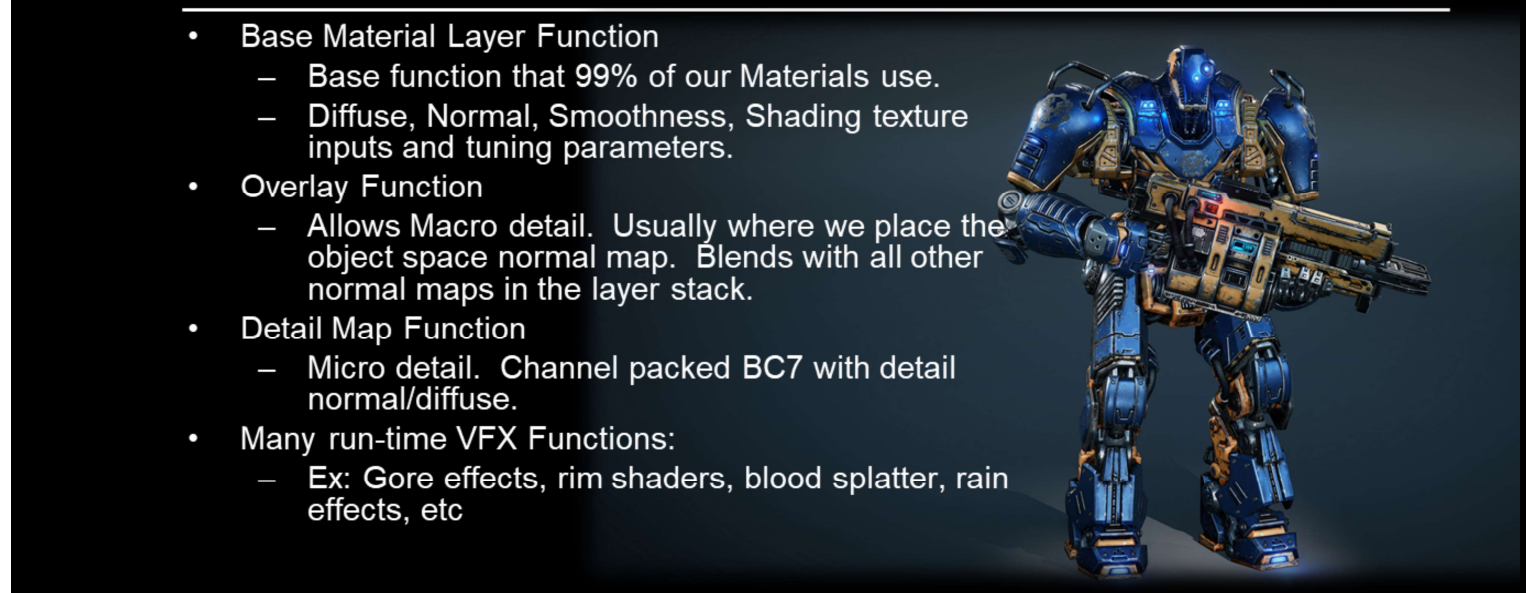
# BLEND MODES

- Each layer is assigned a blend mode.
- Can be complex in most cases as they are smelted.
- We expose material parameters for each blend mode to allow full control.
- Allow overrides in the blend modes.




Certain types of layers in the Material Container UI contain a blend mode. We modeled the UI look after Photoshop essentially.
The blend modes are smelted out in the majority of cases so they can have complex blend functions that give these artists the exact response they would like.
We also expose blend mode parameters that allow artists to tune the exact type of blend they will get, as well do overrides if desired to modify the look of a swatch or nested container.
In this example screenshot I'm using a Height Lerp blend mode, and have selected vertex paint channel B to be the blend colour in the blend mode parameters.
We also allow the artist to define if each layer will "smelt" or "not smelt" which you can see to the right of the blend mode.

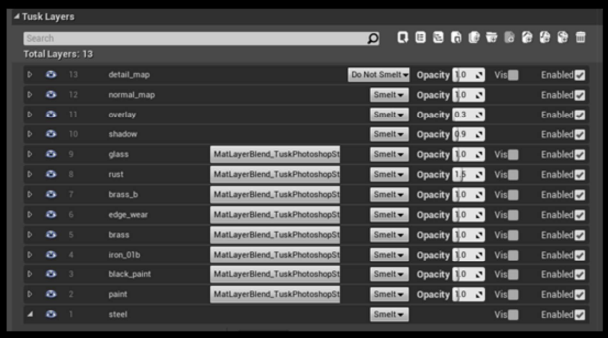
# COMMON FUNCTION EXAMPLES



- Base Material Layer Function
  - Base function that 99% of our Materials use.
  - Diffuse, Normal, Smoothness, Shading texture inputs and tuning parameters.
- Overlay Function
  - Allows Macro detail. Usually where we place the object space normal map. Blends with all other normal maps in the layer stack.
- Detail Map Function
  - Micro detail. Channel packed BC7 with detail normal/diffuse.
- Many run-time VFX Functions:
  - Ex: Gore effects, rim shaders, blood splatter, rain effects, etc

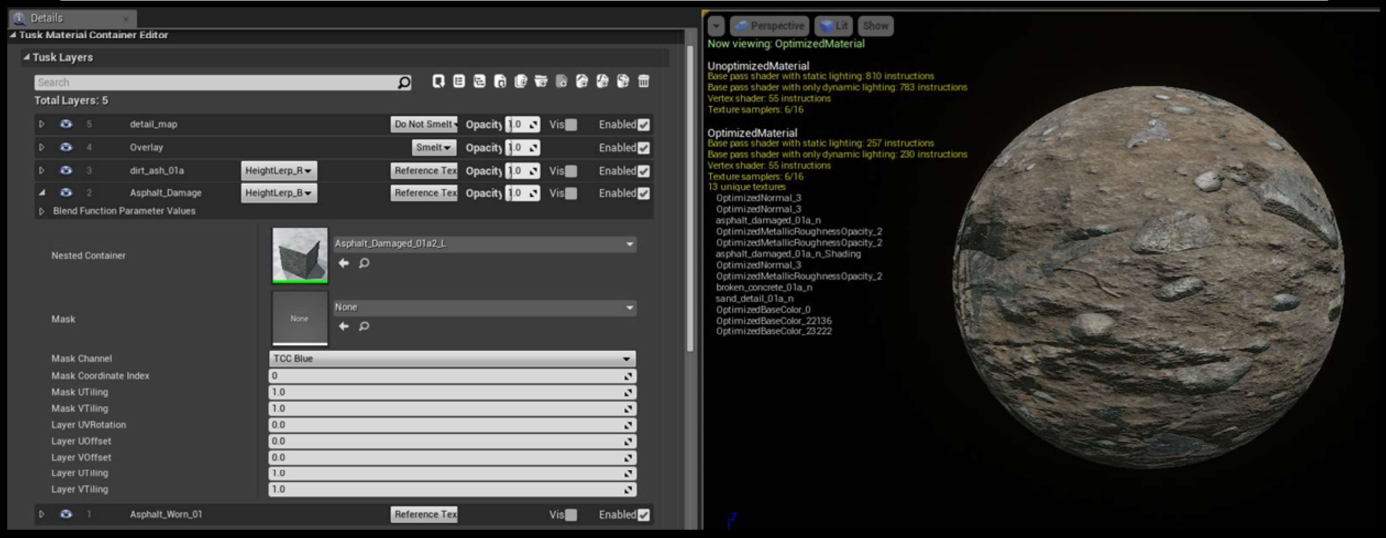
Here are some of our common material functions that we use. These define what functionality a layer will have in our container.
Our Base Material Function is used most of the time and has all the standard controls you might expect. Since we know it will be smelted out we can have some additional complex controls like “contrast” etc.
We have an overlay unction that allows us to apply macro detail. This is usually where we place the object space normal map – which composites with all the swatch normal maps in the stack.
The detail map function is for micro detail. We channel pack a BC7 with normal map and diffuse data in one for efficiency. This layer is almost always set to “do not smelt” so we don’t lose detail on smelting down.
We have created dozens of VFX functions over the years for various effects. Most of these are real-time “do not smelt” layers.

# EXAMPLE ENVIRONMENT ARTIST MATERIAL




Here is an example of what an environment asset container might look like. It contains 13 layers that make up the asset.
You can see at the top the detail map layer which is set to “do not smelt”. This is because we don’t want that detail normal texture to be smelted down with the rest of the normal maps. We also may keep layer as “do not smelt” if they have parameters we would like to update in the run-time such as “team colour”. Also, due to various technical requirements we always want the “run-time” layers at the top of the stack.
Below that you can see the “normal map” layer which contains the object based normal map.
The top 4 layers don’t have an exposed blend mode pull down menu since the blend mode is inherently built into the material function.
Layer 1 to 9 are standard material layers that are using our standard photoshop masked based blend mode.

# EXAMPLE LEVEL ARTIST MATERIAL




Here is an example of a more real-time scenario using one of our ground materials. Most of these layers are set to “reference textures” which means the textures will be reused from another smelted container to minimize disk footprint impact. This container isn’t fully smeltable because we have height lerp blends between different tiling scenarios on this material layers. Instead the smelter cooks the data within each layer individually then live blends it. It’s a really powerful way to work where you can determine all the blend modes, vertex colours, and blending order in your container – then experiment with using different nested material containers inside that stack to get the ideal look for your asset.

# PERFORMANCE

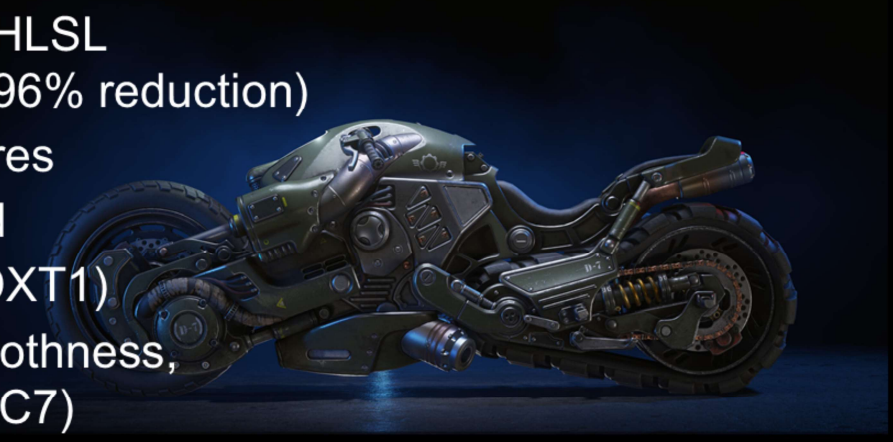
| Full Screen Quad Test on Xbox One - 1 Layer | Info | Milliseconds |
|---|---|---|
| Coalition Material Masking System | 1 layer, 25 controls | 1.37 |
| Vanilla Base Material | 1 layer, 5 controls | 1.5 |

| Full Screen Quad Test on Xbox One - 4 Live Layers | Info | Milliseconds |
|---|---|---|
| Coalition Material Masking System | 4 live layers | 1.9 |
| Vanilla Base Material | 4 live layers | 2.9 |

We did numerous full screen quad PIX captures on Xbox One to determine the sweet spot for the system between texture cache bound vs ALU. Each layer here represents a standard material (diffuse, normal, smoothness). Because we cook down all of our parameters to textures it's hard for any vanilla material to compete on performance.
You can see here that we are able to get large amounts of artist controls into our materials in Unreal with no overhead performance cost.
We have added numerous performance controls as well such as allowing artists to trim channels and replace them with constants for if desired.
I'll now hand it over to Ian who will dive deeper into the technology behind MMS.

# MATERIAL OPTIMIZATION

- Optimize shader and textures
  - 3334 -> 119 HLSL instructions (96% reduction)
  - 56 -> 2 textures
- Textures packed
  - BaseColor (DXT1)
  - Normal, Smoothness, Metalness (BC7)



- To achieve these levels of performance, MMS includes a material optimizer.
- This process takes the artist-created material and outputs a new material with supporting textures.
- As an example of what we were able to achieve, our most highly optimized shader went from 3 thousand, 3 hundred, thirty-four HLSL instructions down to 119, and 56 textures down to 2.
- That's a 96 percent reduction, in the best case.  On average, we reduced materials by 68%, landing at about 150 instructions.
- We also pack texture output in a specific way. The two most-used textures are BaseColor and our Normal, Smoothness, Metalness map.

# SMELTING A MATERIAL

- Analyze the shader and account for artist hints
- Optimized material indistinguishable from the original (usually)
- Material layer granularity
- 2 passes: Layer grouping, layer-by-layer
  - Smelt a group of contiguous layers together if they are not dynamic
  - Then smelt individual layers that could not be grouped
- A great place to generate additional data!

So the artist works initially with what we call the Editor Material, which allows realtime editing. And the optimized material is what we call the Smelted Material.

The smelter's goal is to remove all instructions and data that are not needed at runtime.

To do that, it analyzes the shader graph so that the optimized shader is in-distinguishable from the original. Usually.

So, how does the smelter work?

- We analyze the material at material layer granularity.
- And we perform two passes. The first pass attempts to group layers together. As long as there are no dynamic pieces found in a layer or its blend function, contiguous layers can be smelted together. This produces one material fragment along with a set of textures.
- The second pass smelts **individual** layers that **could not** be grouped. Each of these layers also smelts down to An Optimized material fragment and set of supporting textures.
- All the fragments are then combined to create the final optimized material.
- And a special note **here** is that with all of the **data** available to the smelter, this is also a great place to generate additional data. This is something we **really** take advantage of in our next project.

# IMPLICATIONS

- Texture reduction (intra-material)
- Sharing textures (inter-material)
- Many materials reduced to instances of a parent material that differ only in parameter inputs.
  - Allows for draw batching
- Physical materials (for audio, VFX, etc.) that match the visual surfaces

Implications.

Within a material, we saw a large reduction in textures. But the ability to nest a material within another also allows for texture sharing between materials.
This type of sharing **also** results in overall texture reduction, but it must be balanced with performance. With more texture sharing, you have more texture taps, and layer blends become live. Adjusting this balance is one of the controls that we had when tuning for performance.

The smelter is often able to reduce shaders down to an instance of a parent shader. These shaders run exactly the same microcode. They differ only in parameter inputs.
In Gears 4, the most common parent material was about 150 HLSL instructions. Roughly 60% of our materials were instances of this parent.

So now we have beautiful, high performance materials. But for multi-layered materials to **play** right, the physical materials associated with them need to match what you see. **We** created a process that updates mesh instances so that the correct **physical** materials could be retrieved at runtime. This is one example of the extra data **available** in MMS that we take advantage of.

# MAINTENANCE

- Continuous update
  - Ensures all materials are updated and smelted
- Continuous stats
  - Also compiles materials as it goes, populating the DDC (Derived Data Cache)
- Physical material cooker

By the end of the initial release of Gears 4, we had a library of roughly 8000 inter-dependent materials. Our DLC pushes that number up to 15,000 and it's still growing.

Because of dependencies, materials can become unsmelted. So, for example, if someone updates a texture that affects a **metal** material, and **that** metal is nested in **other** materials, their **smelted** materials are no longer valid. In this case, we revert to the **unoptimized** version of the material. Since an expensive material could take 10ms or more to draw in its **un**smelted form, we needed a way to keep materials up to date and smelted.

Our first line of defense was the editor itself. We added change propagation to dependent materials. The editor then tries to smelt and resave the affected materials. But this is not always possible, for example, if the file is locked.

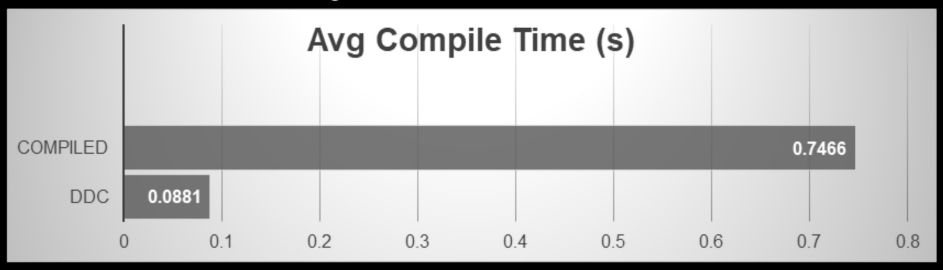
So the second line of defence was our continuous update process, which would go through our library and update materials as needed.
We also had a continuous stats gathering process which we used to monitor the material library. This had the added benefit of **compiling** all the materials, which helped to populate the DDC. The DDC, if you're not familiar with it, is Unreal's derived data cache, which is a shared cache for compiled

resources.

We **also** had a physical material processor, to keep our physical materials accurate. And this worked similarly to our light baking process.

# SHADER COMPILER IMPROVEMENTS

- Multiple compile queues. Local and Distributed. Latency vs bandwidth.
- Compile cancellation for all compile queues. Important because artists make many changes in succession.
- Shader DDC keyed on actual shader source.




- Your artists will be making **lots** of changes to materials, and they'll expect to see the results in realtime.
- Some of the changes they'll make are just to parameters. Those don't require shader re-compilation. But many types of edits require the shader map to be regenerated. A material's supporting shader map can contain a few **hundred** shaders and each one takes in the ballpark of a second to compile.
- To mitigate this, we implemented multiple compile queues. One queue on the local machine for low latency results, and one distributed queue for large jobs such as those that are generated by the change propagation system.
- Since artists tend to make many changes in succession, we also implemented cancellation for all of our compile queues, so that we didn't spend time compiling shaders that were not going to be used.
- And, in addition to Unreal's shadermap DDC, which caches a complete set of shaders for a material, we also implemented an individual shader DDC, keyed on hlsl source code. Different materials that are structured in the same way can actually generate the **same** hlsl source code. So instead of recompiling, these shaders could be retrieved from our Shader DDC. This graph here shows how long it takes for an individual shader to compile versus one that is retrieved from the DDC. Again, this is just a **single** shader permutation. So the time savings for a full shader map are really substantial. It makes edits like undo- and redo- incredibly fast.

# SUMMARY: CHALLENGES

- Took a few years for artists to extract the full power of the system.
  - Both technical and team culture obstacles
- Keeping artists happy. Invest in UI, and education.
- Highly interdependent system
- Compile times of more complex materials can be a problem with less powerful machines (OS vendors)
  - Currently limited to about 30 layers before compile times become detrimental. Most assets 15 to 25 layers.

- I'd like to summarize some of challenges and wins that we got from MMS. First, the Challenges.
- It took a few years for us to fully utilize the system. This was due to both technical and team culture issues.

- If you build a system like MMS, your artists will need time to adjust to the new way of thinking and working. That's hard, so keep them happy. **Make** your UI fast and informative. **Education** goes a long way. So **partner** engineering with tech art and ensure your artists get proper training.

- Materials are dependent on Textures, Material Functions, and other Nested Materials. With over 8000 materials, keeping everything running smoothly was not easy. Our continuous smelter and stats processes were essential in making this workable.

- We invested a lot of time into speeding up shader compilation. Without that work, we would have had very **un**happy artists. We currently cap our materials to about 30 layers which is roughly 3000 hlsl instructions.

# SUMMARY: WHAT WORKED

- MMS is a framework where artists reuse pre-audited shader fragments and embed each others materials
  - Speeds up material creation
  - Optimization is handled for the artists (Smelter)
- Flexibility for those who want to explore
- Engineers and Tech Art can tune pipeline, without changing the inputs
- Predictable runtime performance results.

- And now, our successes.

- First of all, for a typical artist, working with a library of pre-approved building blocks is a huge benefit because it **speeds up** material creation. It **removes** the expensive audit-and-rework part of the iteration loop by ensuring a uniform and compliant result across your material library. This is especially true for outsourced content. In short, artists can concentrate on what they are good at.

- As a framework, MMS is surprisingly flexible. Our artists have created awesome, **re-usable**, plug-in features like our heat discoloration, procedural layer.

- As for Engineers and Tech Art, we made tweaks throughout the project. We changed things like texture packing, and our base material function, and the smelter. We could do **all** of this across the **entire** material library, **without** reworking any of the artist inputs. Loosening the ties between output textures and authored textures allowed us to have full control of the pipeline.

- And of course, with the smelter, we could rely on consistent runtime performance.

# SUMMARY: WHAT WORKED

- Everything in Unreal
  - Material library portability.
  - Tuning materials in-engine takes away any guessing games. Integrated with lighting systems.
  - Authoring blend modes and material layers using Unreal's material creation process is a natural way to work versus learning another app.

We moved as much of the pipeline as we could **in**to Unreal.

- This made it very easy to share our material library, for example, with our external partners.
- It was **also, really** beneficial that we could tune our materials in the **same** rendering engine that would run the game. So your actual **rendering** pipeline, your **lighting** systems--they're are all running, showing you exactly what you will see in the final game.
- Our authoring tools were familiar and in one place. Anyone who knew how to use the editor could quickly get up to speed on materials. There's no need to jump back and forth between apps.

And finally, we're really happy with the results. The visuals, speak for themselves.

Now, I'd like to ask Colin back to wrap things up.

# LEARNINGS

- Went through many iterations
  - Both technical and philosophical.
    - MMS 1.0 = Focused on PBR compliance over ease of use in some cases.
    - MMS 2.0 = Focus on nesting of containers. Razor focused on workflow.
- UE4's workflow quality is so high that any modification to the engine will be compared to baseline workflow quality.
- Getting artists bought in is the hardest part
  - Changing workflows too drastically can result in an artist revolt.
  - Focus on usability first, then enforcement second.
  - Systems like this requires artists to think more technically. Ensure your team is ready for it.

This was a really massive learning experience building up a system like this. In all honesty - when we first rolled out version 1.0 of this system the artists didn't quite get it. They didn't understand the benefits it provided and liked their current ways of working, primarily in Photoshop. We believed in the vision we set out - but began to make a lot of workflow modifications to the system to allow the artists to work the way they wanted. Ironically the mms system today is used very similarly to the way we initially pitched the system years ago. The artists now love the system and the new way of working – and I think the system is a lot stronger because of all the artist input as well as tech art and engineering guidance. For me the takeaway was to always stay customer focused throughout, balancing that with being able to push artists slowly out of their comfort zone if there's something you believe in strongly. It's not easy with large teams and multiple external partners, but it was worth it and I'm really proud of the results.

# NEXT STEPS

- Artist defined UI elements for material containers
- Painting texture masks in Unreal
- Deeper Substance Integration
- Less reliance on textures – more runtime generated detail
- Continue to optimize and tune materials and output
- Generate other useful data during smelting
- Crossover into other domains
  - Eg. Assembling vertex shaders

We have lots of ideas where we can take the system next. Since the core workflow blocks are in-place - most of our time will be spent refining what's already there.

In the future we may try to bite off something larger like mask painting in Unreal – this might be possible with some of the newer render to texture systems in UE4 perhaps.

In some ways, materials are getting more complicated in our industry, with tessellation shaders, complex vertex shaders, and various other high end techniques becoming common - so having the ability to store all that on a material in the database that can be reused and shared is very convenient. To that end - I think we will be spending some time adding more functionality to the system to support more high end features in the near-term.

# THANKS!

- Special thanks to those who contributed to the system:
  - Cindy Wong
  - Anthony Marraffa
  - Ryan DowlingSoka
  - Brad Sweder
  - Malcolm Andrieshyn
  - Eugene Slautin
  - Colin Matisz
  - Kurt Diegert
  - Cedric Lee
  - Phil Cousins
  - Jim Malmros
  - Mike Rayner
  - The Coalition studio for making an awesome looking game with the system.
  - Splash Damage and our other external partners.
  - Epic Games for making an excellent toolset and for their support on Gears.

A big thank you to all the people on this list that have contributed to the Material Masking System. It's been a large multi-discipline effort.
We'd like to thank The Coalition for making an amazing looking game using MMS.
We'd also like to thank Splash Damage and our other external partners for their contributions to the system.
Finally we'd like to thank Epic games for making an excellent toolset with Unreal Engine 4 that inspired us to take a risk and build something new.

# REFERENCES


- David Neubelt and Matt Pettineo 2013. Crafting a Next-Gen Material Pipeline for The Order: 1886. ACM SIGGRAPH 2013 Course Notes. (July 2013). DOI: https://doi.org/10.1145/2776880.2787670

# FULL TIME POSITIONS

http://thecoalitionstudio.com/#Join-Us

Contact: Andrew Glover, andrewgl@microsoft.com

- Lead Character Artist
- Gameplay Engineer
- Rendering Engineer

# Contract POSITIONS

Contact: Jack Coleman, jcoleman@aquent.com

- Concept Artist
- Cinematic Lighting Artist
- Gameplay Animator
- Graphic Designer
- Layout Artist - Cinematic
- Lighting Artist
- Level Artist (Intermediate & Junior)
- Level Designer
- QA tester
- Technical Director - VFX
- Tools Software Engineer
- VFX Artist
- UI Engineer