

GDC 2025

# Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

Martin Palko - Lead Technical Artist

1

Title Slide  
Revolutionizing Texture Pipelines:  
EA's Journey with Texture Sets

**Presenter**

My Face



**Martin Palko**

Lead Technical Artist @ SEED

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

2

I'm Martin Palko, Lead Technical Artist at SEED.





A bit about me:

- Got my start making maps and mods for Unreal and C&C games.
  - Previously at Crytek, working on VR titles and Hunt Showdown, as well as Eidos Montreal for SOTTR
  - Joined Motive 6 1/2 years ago to work on squadrons, and was the lead tech artist for dead space.
  - I then spent some time on Motive's next project, before joining SEED to do R&D
- So what is SEED?



# Search for Extraordinary Experiences Division

[seed.ea.com](http://seed.ea.com)

Short for the "Search for Extraordinary Experiences Division"  
and is electronic arts' central, applied research division.  
But what am I going to talk about today?

## Intro

### Overview

- A story about Textures
- Problem, and evolution of Texture Sets
- Deep dive
- Current state & the future



### Presentation overview

As you probably got from the title of the presentation, we're going to talk about textures. And specifically, some tech that we've developed over the course of the last few years that we call texture sets.

- I want to set the stage with a quick history lesson, telling a story about textures
- Then I'll talk about the problems we faced, how that led us to design a solution, and then come along for the ride as we iterate and evolved through two different implementations, on two different engines.
- I'll do a deep dive into our latest implementation,
- and then finish up talking about where we see it going in the future



## Intro

### Overview



- Different engines use different terminology
  - Shader (graph) / Material (graph)
  - Data Compilation / Build Pipeline / Cooking
  - Runtime Data / Derived data
- A lot of techniques mentioned

I'd also like to make a quick note on terminology - I'll be talking about different engines in this presentation, and they have different terms for similar concepts.

Specifically relevant will be:

- Shader (graph) / Material (graph): Node based graph that generates HLSL shaders. It's then applied to objects to renders things.

Also Shader preset / Material Instance: A preset of a material graph with different input parameters

Data Compilation/ Pipeline, or Cooking : when Engine consumes intermediate data, to create data optimized for the game, which we then call runtime, or derived data.

- I'll also be mentioning a ton of different rendering techniques in passing. If you're not familiar with the specific technique, don't get caught up on it, as it'll usually just be used as an example, and isn't core to the tech.



# Chapter 1: A Story About Textures

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

7

Chapter 1: A story about textures

Let's tell an (approximate) story about how the usage of texture in games evolved.

## A Story About Textures

### Sprites

- In the beginning, there were sprites



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

8

- In the beginning there were sprites
- Sprites were encoded in various formats, but generally wrote pixel information directly to the screen; we called this blitting

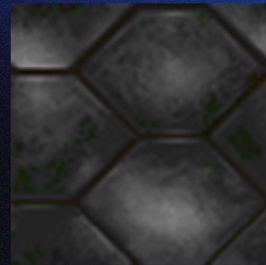


## A Story About Textures

### '3D' Sprites



- What if we moved, scaled, skewed, and warped these sprites to make things look 3d?



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

9

- What if we, scaled, skewed, and warped these sprites to make things look 3d?
- Could achieve a pseudo-3d look, with only 2d images

## A Story About Textures

### Triangles and Textures

- Now we can render triangles!



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

10

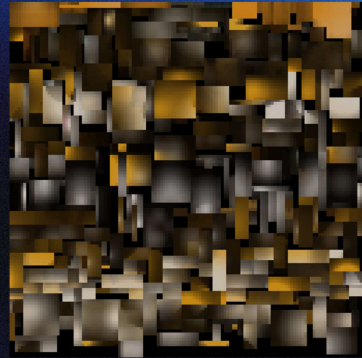
- Eventually hardware caught up, and we could render texture mapped triangles
- At this point, they are mostly color image textures, with an optional 'alpha' channel for transparency - used for things like fences or trees in this scene

## A Story About Textures

### Triangles and Textures



- Modulate textures



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

11

- Things continued to evolve, and some other types of texture maps came onto the scene.
- Notably, lightmaps provided a mechanism for apply pre-baked lighting, and emissive maps to make things glow
- Everything is still mostly RGB colors



## A Story About Textures

Fancy Things



- Let's do some better lighting



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

12

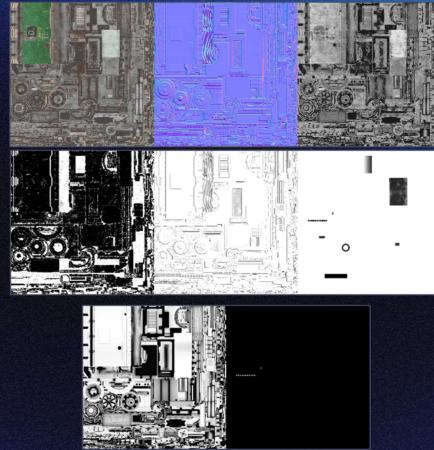
- Then we started pushing lighting and texture details
- We used normal or bump maps to define surface variation
- Specular and gloss maps to control the reflection of light
- Detail maps tiled across surfaces to increase apparent texel density
- Notably, this is when we start having standard textures that are not 'images', but maps that represent a different type of data, yet still layered ontop of an underlying image

## A Story About Textures

Let's get Physical



- Move to PBR



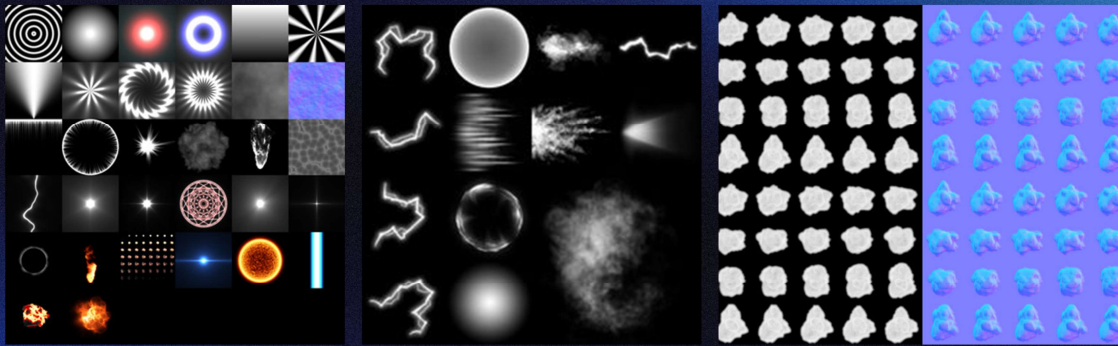
Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

13

- Then we moved to PBR, which is where we are now
- we specify a BRDF:
  - consists of Base Color + Metal, or alternatively Albedo + Specular
  - microsurface is specified in Roughness / Smoothness
- can also add more lighting data: Cavity + AO maps
- These are now entirely material properties, and are no longer technically images
- They also generally share the same UV layout, so we can think of them as a texture with many named channels

## A Story About Textures

And VFX!



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

14

At the same time, vfx evolved to use textures for many different purposes:

- Animated flipbooks
- Noises
- Masks
- Shapes
- prebaked lighting

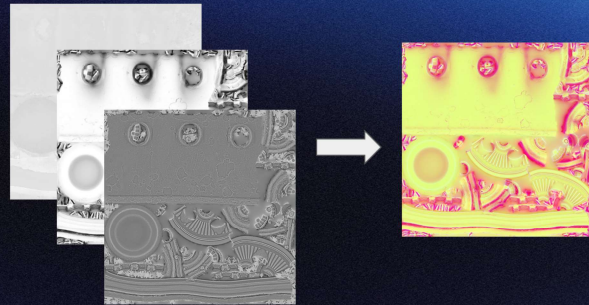
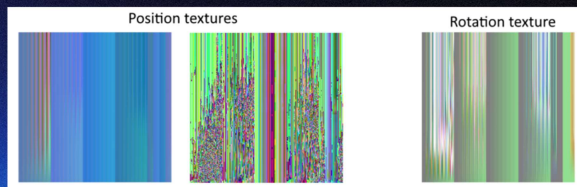


## A Story About Textures

It's just data after all



- These are too big, and have too many texture samples!
  - Channel packing
- This is just a data buffer, right?
  - Vertex animations



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

15

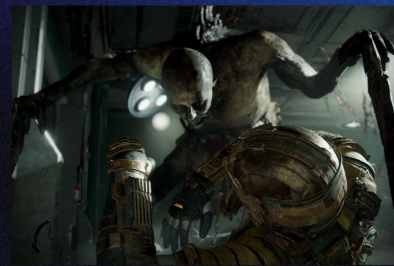
- Also realized that having this many textures took up a lot of memory, and texture samples.
- Our compression formats give us (up to) 4 channels, let's smush stuff together: Called that channel packing
- This is just arbitrary data, right?
  - Stored things like vertex animations in texture maps. Each row could be a vertex, and each column is that vertex's position over time, with xyz packed into the rgb channels.

## A Story About Textures

In Summary



- Went from a texture essentially being 1:1 to a material...
- ...to a half-dozen (or more) textures



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

16

So to summarize: we've gone from a texture essentially being 1:1 to a material, to a half-dozen (or more) textures to fully describe a material.

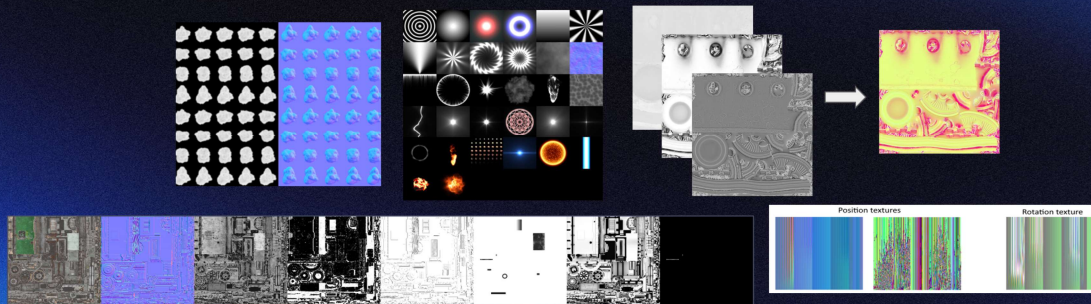
- Lots of different types of textures, used for all sorts of things
- Many compression formats, and the hardware limitations of 4 channels per pixel, are still based on assumptions made before textures were being used this way.

# A Story About Textures

## In Summary



- Lots of different types of textures
- Compression techniques and hardware limitations still based on old assumptions
- Rarely actual images
- DCCs have started to adapt, most game engines have not



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

17

- Rarely are these textures actually images, but we've become accustomed to viewing non-image data through tools designed for images such as photoshop
  - Some DCCs have adapted, or have been designed with this in mind -such as substance painter
  - Yet *most* game engines have not significantly changed how they handle textures
- pause





## Chapter 2: Squadrons

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

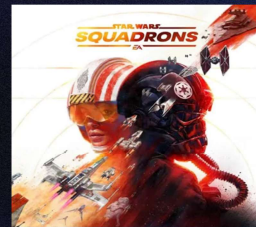
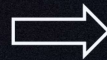
18

Chapter 2 Squadrons: Our journey begins

## Squadrons



- Squadrons forked from Battlefront 2



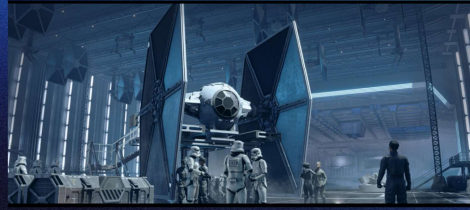
Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

19

- I joined Motive ~½ way through the project that would become Squadrons, just after vertical slice
- Before I joined the studio, motive worked on Battlefront 2's singleplayer. Since the team was familiar with the tech, and was planning to make a similar game, the squadrons project was forked from shipping branch of Battlefront 2
- That meant we initially inherited a lot of code, shaders, and content from the previous project

## Squadrons

- Similar but different
  - Flight & standing sim
  - Cockpit detail
  - Damage & destruction
  - PSVR (Base PS4) to high-end PC



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets



20

It was similar but different

- All gameplay takes place inside the cockpit, or standing in the player's capital ship and interacting with NPCs
  - we wanted to push dynamic damage & destruction on enemy ships as core gameplay
  - we also pushed cockpit detail since for the majority of the game they occupy 1/3 to 1/2 of the player's view
  - entire game had to run on everything from Base PS4, in PSVR up to a high-end PC
- All this meant that by the time we entered production, the texture packing schemes we inherited from battlefront 2 were no longer ideal for this project



## Squadrons



- Desire to rework packing schemes
- Static switch to support alternate packing schemes
- Maintain support for old content

- We wanted to rework packing schemes but the scope of changes would be large
- Found some solutions to make due such as using static switches in the shader to support alternate packing schemes  
but we still had to maintain support for the old content
- We did work on data patching and doing manual fixups, but many assets never got brought over

## Squadrons

"There has to be a better way"



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

22

This got me thinking...

"There has to be a better way"

## Content Community



- Content communities at EA
  - Focused groups
  - Representatives from studios across the company
  - Meet regularly to share experience and steer initiatives
- Part of the content rendering community

Now, at EA at the time (about 2019), we had just spun up what we were calling 'content communities'

Content communities were groups with specific focuses.

They'd have representatives from studios and central teams across the company who focus on a specific domain, who meet regularly to share their problems, experiences, and work together to steer initiatives.

I was part of the content rendering community, focusing on content creation, rendering, and everything in-between.

In one of our meetings, I brought up the issues I'd been facing on Squadrons, and found it resonated with the other in the community



## Content Community

Back in 2019...



- Brainstormed with content rendering community
  - Easy to make changes any time in production
  - Flexible enough to be a general engine feature
  - Win for everyone

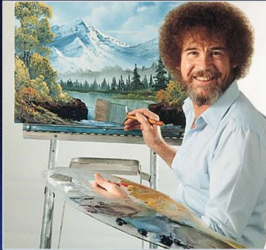
So together we brainstormed what a solution would be.

We wanted something that was:

- Easy to make changes any time in production without requiring mass data patching, or re-importing of content
- Flexible enough to be a general engine feature, not too tailored for a specific project
- needed to be a win for everyone...

## Content Community

Design



### Artists

- Eliminate manual texture packing
- Automatic texture configuration
- Eliminate overhead of managing many textures

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

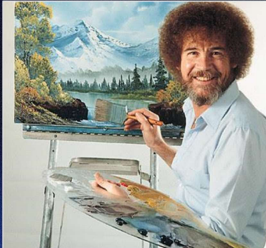
25

### 1. For Artists:

- it would need to eliminate the need for manual texture packing
- Standardized texture settings. so you'd never worry about sRGB settings or the right compression formats again!
- and wanted to remove overhead of managing many individual texture maps (importing, configuring correctly, assigning to materials)

## Content Community

Design



### Artists

- Eliminate manual texture packing
- Automatic texture configuration
- Eliminate overhead of managing many textures



### Shader Artists

- Eliminate manual texture unpacking
- Handle common techniques

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

26

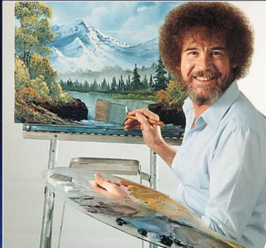
## 2. For Shader/Material Artists:

- remove need to swizzle channels to undo texture packing in the graph.
- Handle common, well-defined techniques such as flipbooking or parallax occlusion mapping – often difficult or messy to properly support in a material graph.



## Content Community

Design



### Artists

- Eliminate manual texture packing
- Automatic texture configuration
- Eliminate overhead of managing many textures



### Shader Artists

- Eliminate manual texture unpacking
- Handle common techniques



### Art Leads / Tech Artists

- Project wide control
- Asset portability

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

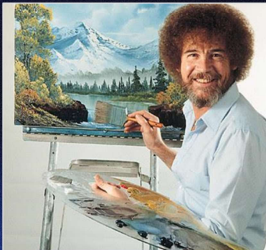
27

### 3. For Art Leads and Technical Artists:

- project-wide control of texture packing layouts and compression formats, allowing iteration and optimization later in production without the need for data-patching
- increase asset portability; allowing importing of assets from external sources and to bring them in-line with project standards without needing to alter the source data.

## Content Community

Design



### Artists

- Eliminate manual texture packing
- Automatic texture configuration
- Eliminate overhead of managing many textures



### Shader Artists

- Eliminate manual texture unpacking
- Handle common techniques



### Art Leads / Tech Artists

- Project wide control
- Asset portability



### Engineers

- Optimized implementations
- Complex techniques

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

28

#### 4. For Engineers

- create a mechanism to Provide optimized implementations of common, complex material techniques in code, which are easily accessible to shader artists. e.g. triplanar mapping, flipbooking with motion vectors, parallax occlusion mapping.

- Allows engineers to author techniques once in a more powerful code generator/hlsl workflow, not have to chase down multiple instances in shader graphs across the project

- we also want to provide a framework for advanced compression or rendering techniques that require both pre-processing and runtime logic

- for example NDF pre-filtering, relaxed cone-step mapping, or derivative mapping which are traditionally cumbersome or impractical to implement at a production scale



### Texture Sets

We have a slack channel! Check out [#texture-sets](#)

▲ This doc was created as a proposal, texture sets is now being integrated into devkit, and supports shader graph. Expression shader support will follow. Check out [modernizing the texture pipeline with texture sets](#) for an overview, and a look at texture sets in action!

#### Overview

Every time, various texture maps such as base-color, smoothness, normal maps, metalness, etc are authored together, but then have to be imported separately and are treated as individual assets in the pipeline/data engine. In addition, texture packing is used to reduce memory usage and texture samples, but this has to be defined by the technical artist in the surface shader. All users of that shader must then manually pack their textures prior to import. This leads to a system that is very rigid. Changing texture packing schemes mid-project ranges from being a huge headache, to near impossible.

The goal of 'Texture Sets' would be to bundle all the texture maps that define a material into a single asset, and allow for the packing, compression, and unpacking to happen automatically.

For **Artists**, it would:

- Eliminate the need for manual texture packing. Just import your texture maps, and the packing happens behind the scenes.
- Reduce the number of assets that need to be managed.

For **Technical Artists**, it would:

- Provide project wide control of texture packing (applies not only texture formats (which we have now with texture groups).
- Eliminate the need for manual texture unpacking in shaders.

In addition, it would open the door to things that are currently not possible, or are cumbersome:

- Automatic build time texture processing (such as HDR texture filtering).
- Optimizing texture compression based on contents (Permutation Normal Map Compression, as detailed in the [tutorial presentation](#) (pg 112)).
- Allow variations of packing (applies per platform, improving scalability: higher quality textures on 'high-end', aggressive packing on 'low-end').
- Allow artist to author with alternative PBR workflows (metal/roughness, spec/gloss) and convert to the working format of the engine in the pipeline.

Other Pros:

- Provides a very compatible path for a substance integration (!) substance = 1 texture set).
- Opt in workflow. Wouldn't break any existing data (some textures would still exist).
- Increase asset portability. Projects could use different packing schemes, and texture sets would be automatically re-packed behind the scenes.

#### Implementation

An implementation would consist of 3 main user facing components:

- Texture Set Definition**: Asset that defines inputs and outputs, packing scheme, and compression formats.
- Texture Set**: The texture set asset itself. Imported from a series of source texture files, a PBR, or (in future) a substance file. References a **Texture Set Definition** to define the required input maps.
- Texture Set Samples**: In the shader graph, a new type of texture node, similar to an existing Texture Sample. References a **Texture Set Definition**. Inputs are the same as a texture node (sample coordinates), but the outputs are the unpacked sample values, as defined by the Texture Set Definition.

#### Texture Set Definition

Texture Set Definitions define the 'template' that imported texture sets, and the texture set samplers will use. They will need to define:

- Input/Output maps
- Packing schemes
- Texture and Streaming Groups (for each packed texture)

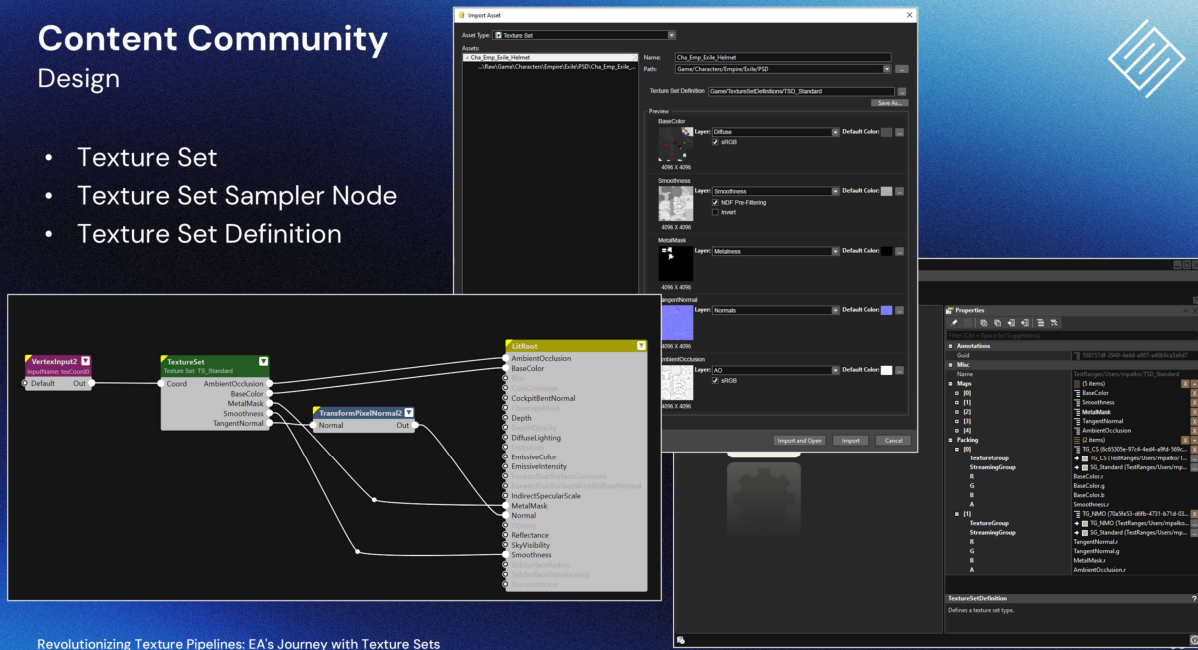
Mockup of a Texture Set Definition

so all this resulted in a document that distilled our goals, as well as a basic design



## Content Community Design

- Texture Set
- Texture Set Sampler Node
- Texture Set Definition



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

Pictured here are the initial mockups of what we planned the tooling to look like.

We'd have 3 main user facing components

1. Texture Set Asset: that contains all closely related textures, and additional metadata
2. Texture Set Sampler Node: which is like a texture node in the shader graph, but instead of sampling a single texture, it samples all textures within a set. The output pins would be named values corresponding to the source elements in the texture set.
3. Texture Set Definition: Data asset that allows the definition of standardized layouts across the project and is used to tie together the texture set assets and the sampler nodes. We could have as many definitions in a project as make sense, and as long as the asset and sampler nodes were using the same definition, everything would be kept in sync.

## Content Community

### Next Steps?

- Can we get Frostbite to build this?
- Maybe some teams want to build it together?

now we've got a good idea of what it is we want to build, how do we get it done?

- Can we get Frostbite to build this?

- They like the concept, but unfortunately it doesn't fit on the roadmap for the next few quarters :(

- Maybe some teams want to build it together?

- Lots of teams showing interest, but none are currently at the "sweet spot" in the project to make it happen

- when I say sweet spot, I mean that some projects were too early, where they didn't have enough resources to develop a piece of software of this scope

- others were too far into production and had too much existing data and processes in place for a large overhaul like this

- So for now, it was put on ice and I got back to Squadrons knowing that yes, there is a better way, but it's unfortunately not for now.



## Chapter 3: Dead Space

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

32

### Chapter 3: Dead Space



## Dead Space

### Context

- Next project for Motive: remake Dead Space
- Unfamiliar performance characteristics



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

33

- Next project for Motive was a remake Dead Space on Frostbite
- As the game is a remake, we have a good idea of the scope of content from the start. We hadn't made a similar game at the studio, and were on a relatively short time frame, so we needed to go wide, quickly.
- Team is familiar with the engine, and rendering wise, it's is relatively well suited for this type of game
- But this is the first game we'd be making for PS5 and xbox series. We weren't yet familiar with performance characteristics of the new console generation.

## Dead Space

### Context

- Texture Sets would be a good fit
- Idea: Get a barebones version up and running asap
  - Author content early
  - Add features in parallel with content



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

34

Seems like Texture Sets would be a good fit

It would allow us to make changes to textures across the project *after* we have had a chance to learn more about the hardware and content without delaying the start of content production.

Idea was:

- Let's get a barebones version up and running asap
- Use it to author content early
- Add features in parallel with content production

## Dead Space

### Context

- Working with the art team
  - Initial guidance:
    - No packing
    - Don't worry
  - Required building trust and showing our work



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

35

Working with the art team

Initial guidance: - don't worry about packing

- No, seriously don't worry about packing. You'll give yourself more work.

- Also memory usage, also perf, don't worry

- We'll fix it, promise

Nerve wracking for the production, and required building trust and showing our work



## Dead Space



- Get to building...



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

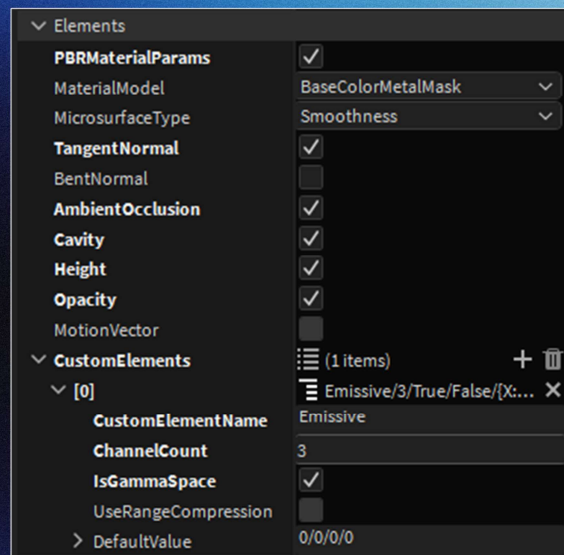
36

Get to building...  
(pause)

## Dead Space

### Implementation

- Texture Set Definition
  - Data asset
  - Standard elements
  - Custom elements



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

37

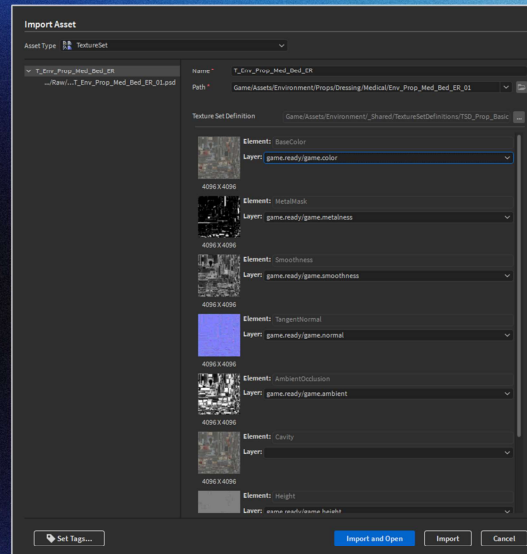
First, we built the Texture Set Definition.

- has Checkboxes to enable/disable standard elements: things like PBR parameters, AO, heightmaps, etc.
  - allows for Standardization - no inconsistent naming
  - since we know what type each element is, it allowed for data Pipeline logic - such as NDF prefiltering
- also had Custom elements: for anything considered non-standard
  - user could specify: Name, Channel Count, Default Value, and Encoding
  - came with no special logic in the data pipeline

## Dead Space

### Implementation

- Texture Set Importer
  - Import from PSD as Texture Set
  - Select definition
  - Map layers to each TS element
    - Auto-populated by string matching layer names



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

38

Then to create Texture Sets, we built an Importer

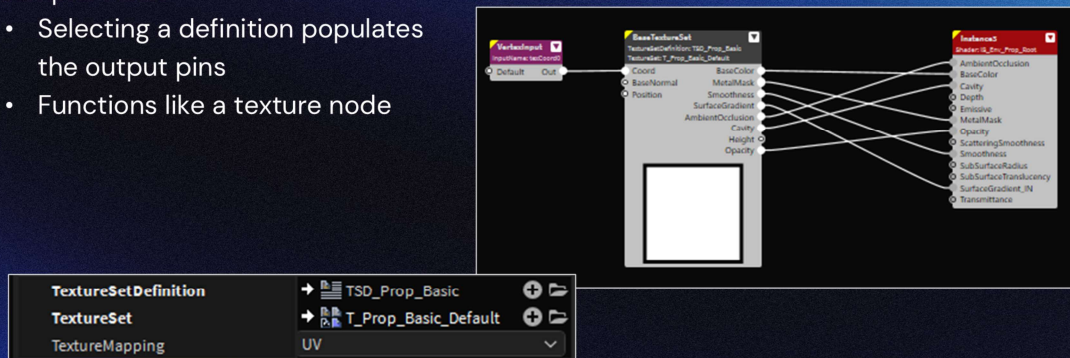
- allows Importing a PSD into the engine as Texture Set
- When importing, you first select definition, which populates the UI with elements as defined in the definition
- then for each element, you select the layer in the PSD that it will read from
- this mapping was auto-populated by matching the psd layer names to a database of known aliases for each element



## Dead Space

### Implementation

- Sampler Node
  - Selecting a definition populates the output pins
  - Functions like a texture node



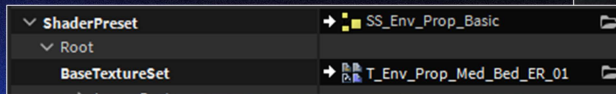
the last piece was the sampler node  
again, the user starts by selecting a definition which populates the output pins  
it then works just like a texture node

- has an input for a texture coordinate
- internally reads the data
- outputs result - this time as named values instead of 4 generic channels

## Dead Space

### Implementation

- Shader Preset
  - Assign just like you would a single texture

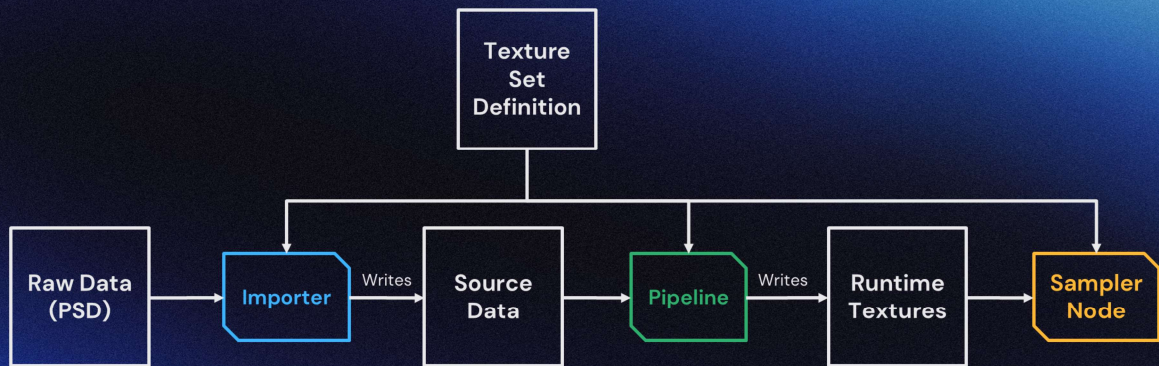


Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

40

on a Shader Preset which is the frostbite equivalent of a material instance in Unreal:  
you assign Texture Set just like you would a single texture

## Dead Space Implementation



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

41

and here's how it worked

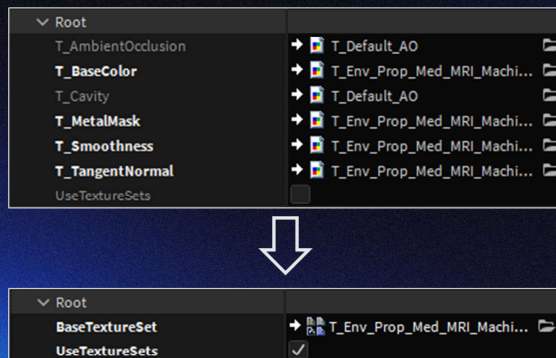
- Definition is a data container. No logic, just referenced by other assets.
- Editor's PSD import extracts target layer as .png into a the asset's source data
- Texture set data pipeline consumes the imported textures, and builds regular engine textures based on the configuration of the definition.
- Sampler node generates shader code based on the definition. At runtime, it binds textures to the material parameters for sampling.



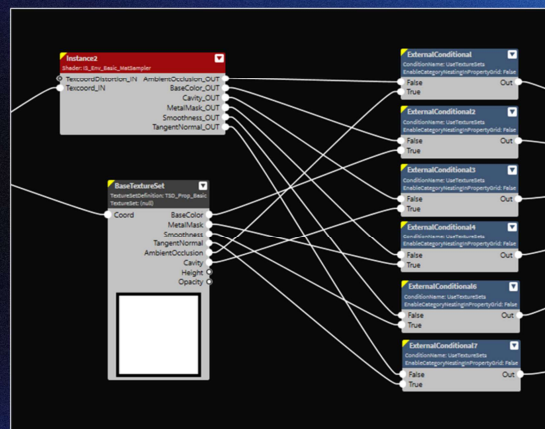
## Dead Space

### Implementation

- Moving the team over
  - Use a static switch in shaders (permutation)
  - Done in phases



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets



With a barebones version now in place, it was time to move the team over.

We did have some content created before this first version of Texture Sets came online, so had to migrate some existing content.

Use a static switch in shaders which created a permutation with, and without texture sets.

The move was then done in phases

- First, we talked to senior artist and art leads on the team until we were confident nothing was obviously broken. It also gave us a chance for a first round of critical feedback
- Then, presented at team meetings, encourage people to explore it.
- Next, we adjusted guidance to favor using Texture Sets as the "right way" of doing things
- After that, we started to lock down usage of texture sets on new assets, but keeping old assets in-tact
- Finally: cleanup - conversion script handled 90%, and then had some manual work to do for stragglers caused primarily by mis-naming of assets.

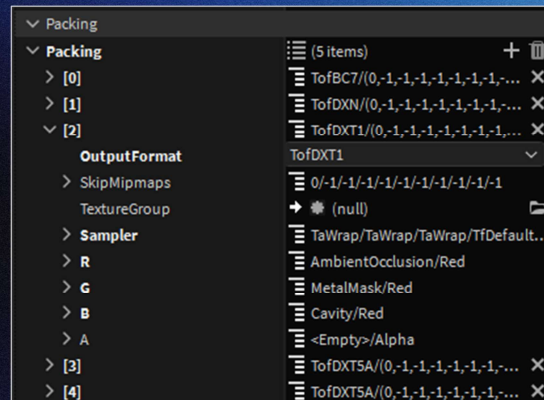
Note: We know all the other benefits texture sets will offer later on. But it's important for smooth adoption that the art team sees immediate workflow wins -- Creating, and working with texture sets should be immediately easier than without.

## Dead Space

### Implementation

- Continue development through production:

- Texture packing
  - Packing definition
- Image Formats
  - Layered EXR
- Other Features
  - NDF prefilter
  - Parallax occlusion mapping
  - Flipbooking



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

43

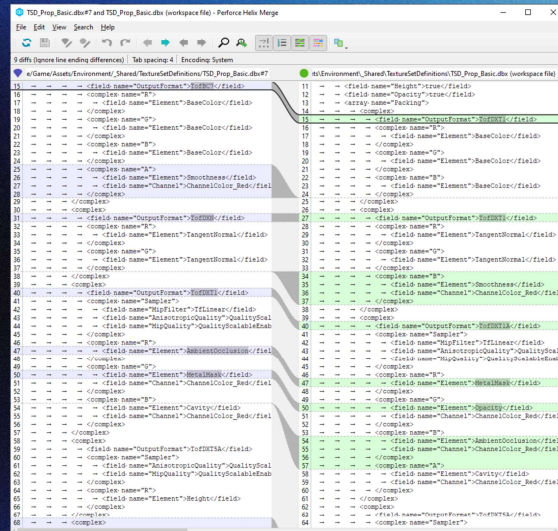
### Continue development through production:

- Texture packing, controlled via *packing* definition on the *texture set* definition
  - Specify a list of packed textures, along with texture formats, and what element resides in each channel
- Added support for other import formats, specifically layered EXRs for handling high-precision source data better than PSDs
- Other Features
  - NDF prefilter (sometimes called normal to smoothness/roughness)
  - Parallax occlusion mapping in the sampler node
  - Flipbooking with frame blending and motion vectors

## Dead Space

### Results

- Enabled large-scale changes late in the project
- Changes to definition was only a submit of a single file
- Iteration times under ~1 hour for a local texture rebuild of a specific level



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

44

So how did it end up?

Framework successfully enabled large-scale changes late in the project

we could add additional textures and features, modifying the packing and texture formats when needed, all without invalidating existing data

It opened a new avenue for optimization, as we could now a-b test different packing schemes and data layouts.

Changes to definition were only a submit of a single file, which was easily reverted  
Frostbite's pipeline and caching handled everything else.

Just to reiterate. What would we have to do without texture sets?

Reimporting/repacking would have had to be done on the raw asset in photoshop or Substance, re-exported, and then re-imported to the engine.

We'd need to update the shader to unpack in the new scheme, add a static switch to not break existing content and then re-assign the new textures to the material.

With thousands and thousands of assets in the project, this would take days to weeks, and been an undertaking to do just once, much less as quick test.

With texture sets, modifying the definition takes a minute or two. And the rest is just build time, taking a few hours to rebuild all the textures in the game.



## Dead Space

### Results



- What would we have done if we had to decide early?
  - Napkin math + educated guess

- Unpacked
- 4.5 Bytes/Pixel - 19mb @ 2k
- 7 Samples

Texture	Format	Byte/Pixel	Channel
0	BC7	1	R: BaseColor.r G: BaseColor.g B: BaseColor.b
1	BC5	1	R: TangentNormal.r G: TangentNormal.g
2	BC4	0.5	Metallic
3	BC4	0.5	Smoothness
4	BC4	0.5	Ambient Occlusion
5	BC4	0.5	Cavity
6	BC4	0.5	Opacity

- Moderate packing
- 3 Bytes/Pixel - 12.5mb @ 2k
- 3 Samples

Texture	Format	Byte/Pixel	Channel
1	BC7	1	R: BaseColor.r G: BaseColor.g B: BaseColor.b A: Opacity
2	BC5	1	R: TangentNormal.r G: TangentNormal.g
3	BC7	1	R: Metallic G: Cavity B: Ambient Occlusion A: Smoothness

- Aggressive packing
- 1.5 Bytes/Pixel - 6mb @ 2k
- 3 Samples

Texture	Format	Byte/Pixel	Channel
1	BC1	0.5	R: BaseColor.r G: BaseColor.g B: BaseColor.b
2	BC1	0.5	R: TangentNormal.r G: TangentNormal.g B: Smoothness
3	BC1	0.5	R: Metallic G: Opacity B: Ambient Occlusion A: Cavity

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

45

What would we have done if we had to decide early?

Would have done napkin math and made an educated guess, based on past experience.

Here's 3 possibilities:

- First for reference, completely unpacked; 7 texture samples, coming in around 19mb for a 2k texture set
- Second, moderate packing; 3 textures samples, and around 13mb at 2k
- Finally, a more aggressive packing scheme, still at 3 samples, but down to 6mb at 2k

We would have chosen probably second or third option, based on how worried we were about quality, size on disk and/or performance.

But now we can do better!

## Dead Space

Results



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

46

Let's take this asset as an example.

We can try out our different packing schemes to see the impact on quality

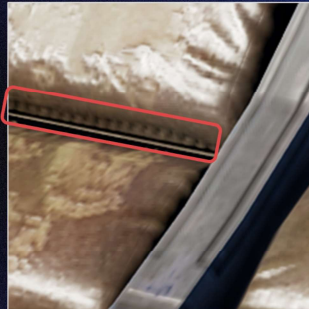
## Dead Space

### Results

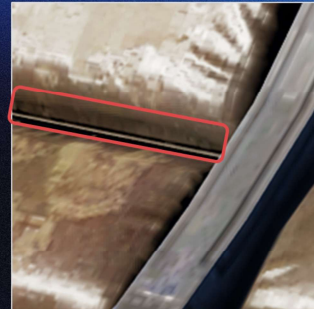
- Unpacked
- 4.5 Bytes/Pixel - 19mb @ 2k
- 7 Samples



- Moderate packing
- 3 Bytes/Pixel - 12.5mb @ 2k
- 3 Samples



- Aggressive packing
- 1.5 Bytes/Pixel - 6mb @ 2k
- 3 Samples



- Could have done this type of test previously. Let's go further

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

47

The moderate packing scheme doesn't lose much detail compared to the unpacked textures, while the aggressive packing starts to get a bit muddy. But wait, you're saying, we could have done this test before. It's easy with one asset; we make 3 version and 3 test shaders, then take some screenshots.



## Dead Space

Results



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

48

Now, with textures sets

- I can easily fire up the editor, 2 years after ship, and in a few minutes re-pack all the textures in this scene.
- So let's test those packing schemes on all the assets in the whole scene, and then benchmark performance.
- modifying the definitions takes a minute or two, and about 1 hour for a local texture rebuild of this level.

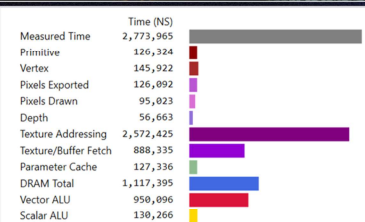
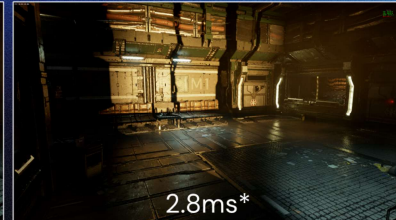
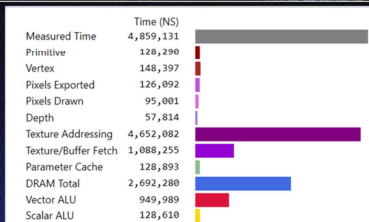
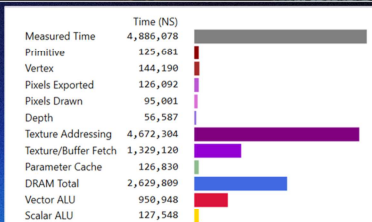
# Dead Space

## Results

- Unpacked
- 4.5 Bytes/Pixel - 19mb @ 2k
- 7 Samples

- Moderate packing
- 3 Bytes/Pixel - 12.5mb @ 2k
- 3 Samples

- Aggressive packing
- 1.5 Bytes/Pixel - 6mb @ 2k
- 3 Samples



\*Gbuffer laydown, 1080p, Series X

49

Here's the result. We're benchmarking just the gbuffer laydown pass, @1080p internal resolution on Series X.

- Unpacked, we run about 5ms, and the aggressive packing sees a great performance improvement, at 2.8ms.
- Surprisingly, the moderate packing comes in at almost the exact same performance as the unpacked textures.
- While we're seeing benefits for memory usage, and size on disk, we aren't getting the performance increase we might expect.
- I was a bit shocked by this, actually. When putting together this test for the presentation, I assumed it would be somewhere in-between.
- This goes to show, educated guesses are often wrong. The only way to be sure is to test!

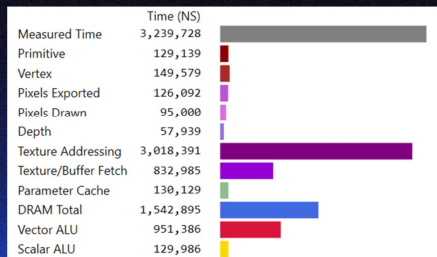
## Dead Space

### Results

- Could do a whole talk on optimization work we did
  - Texture formats & packing schemes
  - Skip mips / lod bias
  - Texture filtering

- 2.5-3.5 Bytes/Pixel
- 10.5mb - 14.5mb @ 2k
- 3-5 Samples

3.2ms



Texture	Format	Byte/Pixel	Channel
1	BC7	1	R: BaseColor.r G: BaseColor.g B: BaseColor.b A: Smoothness
2	BC5	1	R: TangentNormal.r G: TangentNormal.g
3	BC1	0.5	R: Metallic G: Cavity B: Ambient Occlusion
4	BC4	0.5	Opacity
5	BC4	0.5	Height

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

50

Could do a whole talk on optimization work we did, but I'll try to summarize our findings:  
For packing schemes:

- BC7 gives basecolor and smoothness a good/adaptive quality
- Unpacked BC5 normal maps prioritizes quality
- Can get away with BC1 for masks that usually have detail in one location in a channel where there's no detail in other channels. This helps avoid channel 'cross-talk' issues.
- Optional maps like opacity and height are put in their own textures.
  - This lets texture sets sub in a 4x4 default texture when they're not needed
  - Having opacity on it's own also speeds up depth-only passes as we're not sampling a whole 4 channel texture to then discard ¾ of the result.

We used skip mips, sometimes called a lod bias

- Opacity and Height are easy candidates to downscale.
- as well as masks for some asset types
- Found Texture filtering to be just important as packing, and more important than texture size for performance.
  - We biased texture filtering quality per packed texture with higher quality on normals and lower quality on masks
  - In fact, we could go down to bilinear filtering on opacity and height with almost no visible difference



- To reiterate: We only had the time to confirm these things actually worked for our specific context, because we could a-b test entire scenes efficiently.
- This example packing scheme doesn't represent the entirety of assets in the project. Some things like detail textures, or transparent shaders would use different approaches, but again validated through a-b testing.

## Dead Space

### Results



- Positives
  - Massive gains to gbuffer laydown
  - Artist experience was generally positive
    - "When I work on personal projects on the weekend, I miss Texture Sets."
- "Opportunities"
  - Build times
  - Monolithic
- Stats
  - 6,255 Texture Sets
  - 35 Texture Set Definitions
    - Most Popular: 3,504

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

51

Back to overall results:

As mentioned: great gains to gbuffer laydown

After optimization, gbuffer pass performance was rarely a concern (Environment artists rejoice)

Artist experience was generally positive. Got feedback such as:

"When I work on personal projects on the weekend, I miss using Texture Sets.", which obviously made us very happy to hear.

Build times crept up as more was happening in the build pipeline than previously.

Frostbite's equivalents to data caching, and a virtualized asset system helped the process go smoothly. More time could have been spent optimizing the processing itself. Also, the monolithic nature of the pipeline was an issue.

both the Texture processing function: which was a big function that looped through all the texture data with "if this type of texture, do this"

as well Shader generation: which was a big function that wrote out the HLSL code of the sampler node.

- we were 'Architecting' as we go and didn't have a lot of time for cleanup or refactoring

Stats: over six thousand texture sets, using 35 definitions, with the most popular definition used by about 3500 texture sets

## Dead Space

Results

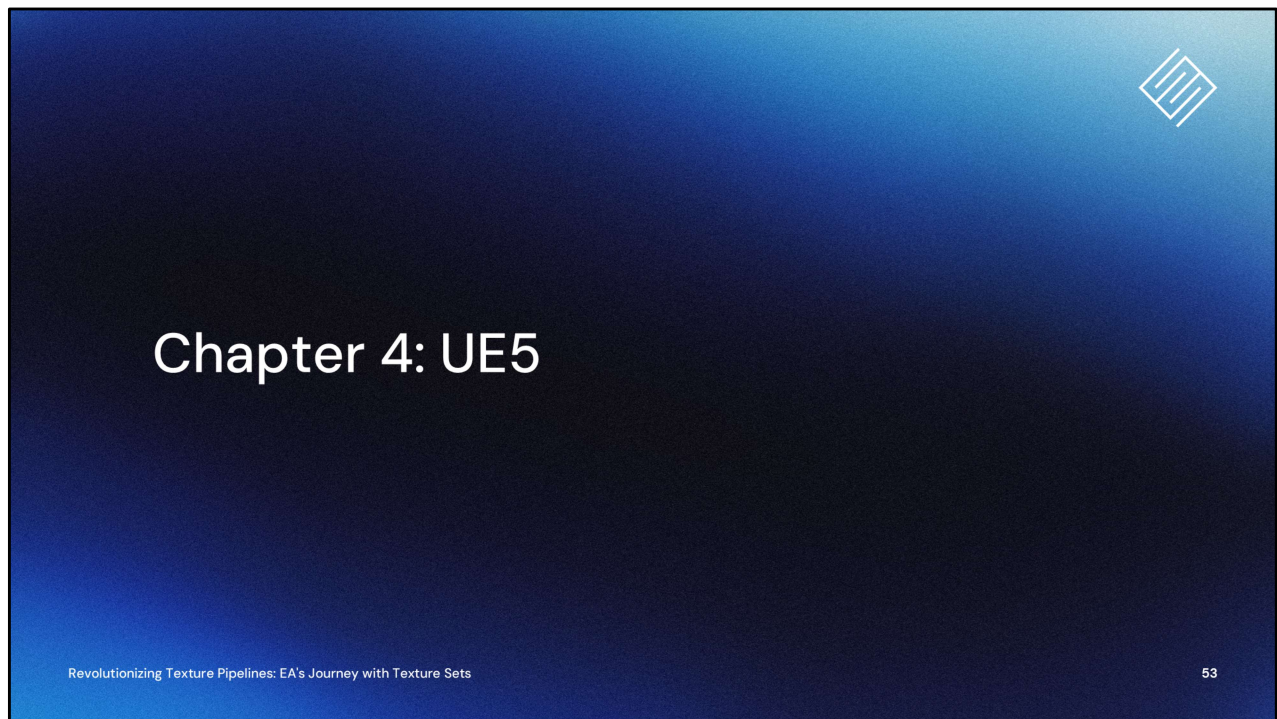


Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

52

But all in all, great success!  
Whats next?





## Chapter 4: UE5 - Starting fresh



Next project for the studio would be on UE5  
Still see the value of texture sets, so let's build it again?

## UE5

### Improvements

- Incremental improvements
- Code architecture and quality
- Efficiency
- Extensibility



Since we're re-implementing, let's take the opportunity to improve

- Small incremental improvements like UI, nomenclature, etc.
- Focus on code architecture and quality.
- Wanted to keep in mind and benchmark Efficiency – namely build times
- But the big thing was: Extensibility – Can we re-architect this to make it extensible?
  - This will be a requirement to continue expanding feature set.
  - as mentioned previously the monolithic code gen and texture processing from our previous implementation got unwieldy.
  - A plus to this would be that future teams could add their own project-specific features



## UE5

### Constraints



- Engine divergences
  - Expensive to maintain
  - Can we build this without diverging?
- Uncharted territory for the team

Project came with it's own set of constraints  
the main one being engine divergences

- expensive to maintain
- Project is early, and the team wants to take engine upgrades with low friction
- Can we build this without diverging the engine?
- If we need to diverge, How do we minimize it? what's the strategy?
- Uncharted territory for the team, as we were new to the engine and tech

## UE5

### Plan of Attack



- Plugin
- Direct artists to author content with unpacked textures
- Confident we can get it going quickly. We've done this before. Right?
  - Not so fast...

Plan of attack:

Author as a plugin

- First, try to keep everything in the plugin, without modifying engine code
- When needed, try to make engine modifications minimal, and as generic

extensibility points, that could make sense to send back to Epic

As before: direct artists to author content with unpacked textures

- Means performance and memory will be problematic until TS is up and running
- but we're confident we can get it going quickly. We've done this before... right?

## UE5

### Plan of Attack

- It was a journey
  - New to UE5
  - Engine changes
  - Data pipelines

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

58

### Not so fast

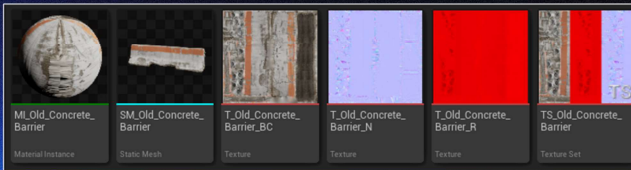
- New to UE5 (myself, and as a team)
  - Didn't have engine experts as readily accessible as when we were using Frostbite
  - Avoiding divergences took a lot of effort. We did still ended up with a few that were unavoidable
  - As I'll touch on later, the UE5 data pipeline was also less well suited for this sort of thing
- Took much more effort than anticipated, It was a journey.  
So what does it look like?



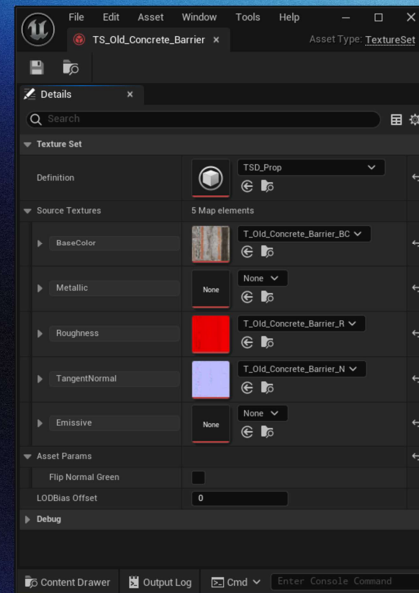
## UE5

### Implementation

- Texture Set
  - References definition + textures in project



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets



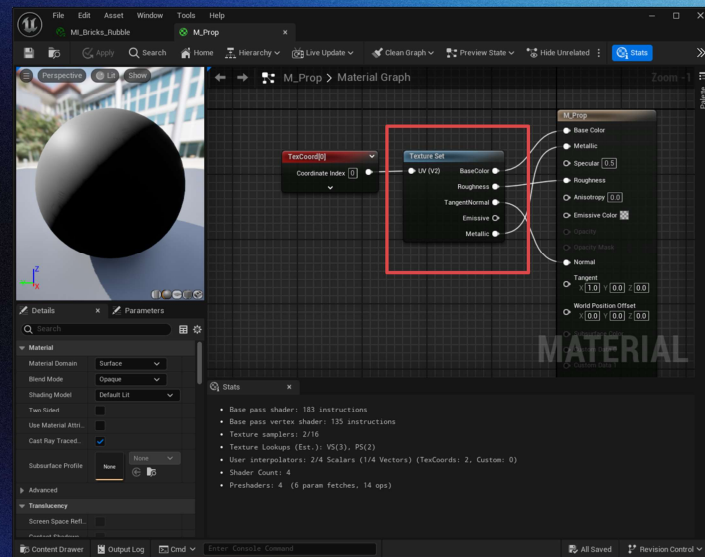
59

### Texture Set

- This is our version of a texture set in UE5
- References Definition, as before
- but we were on 5.0 when we started writing this, and Epic had talked about their intent to overhaul the import process with something new called the interchange pipeline.
- It wasn't ready to use yet, so we decided to work around the import process until it stabilized.
- Instead of importing, we now reference other textures in project for each element.

## UE5 Implementation

- Sampler



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

60

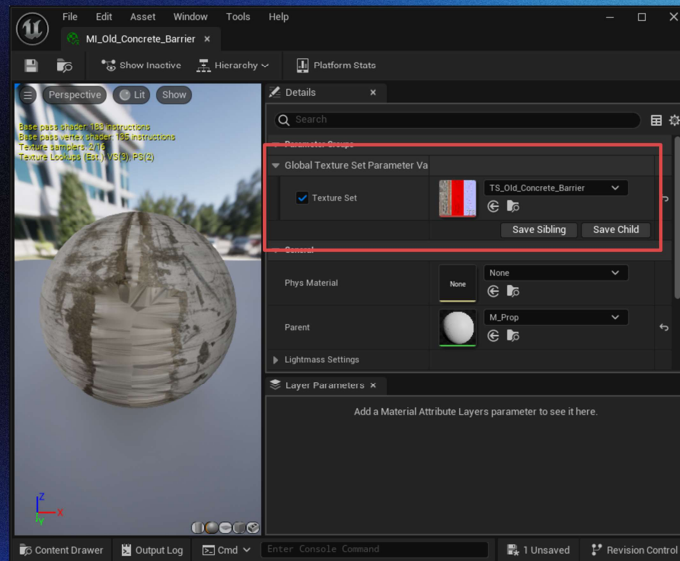
This is the sampler node

Like before, you reference a definition which populates the input and output pins.

The sampler node is also a parameter ...

## UE5 Implementation

- Parameter



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

61

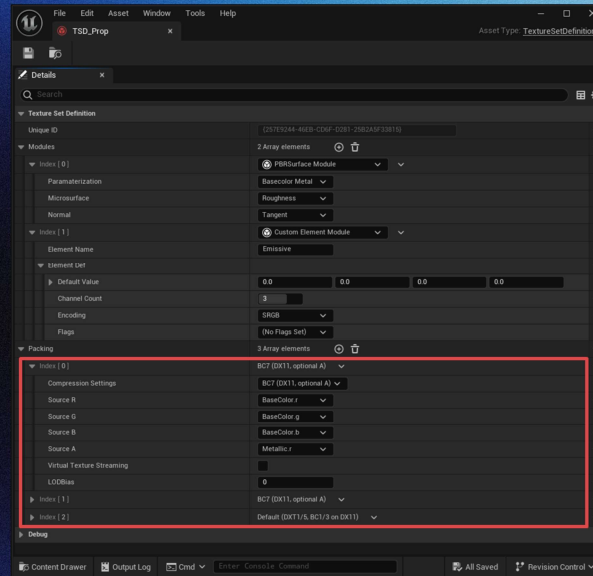
... which creates a texture set parameter on material instances  
This parameter only accepts texture sets which use a matching definition to that of the sampler node, ensuring that the unpacking logic matches the packing layout.



## UE5

### Implementation

- Definition
  - Packing Definition
    - Derived textures
    - Channel contents



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

62

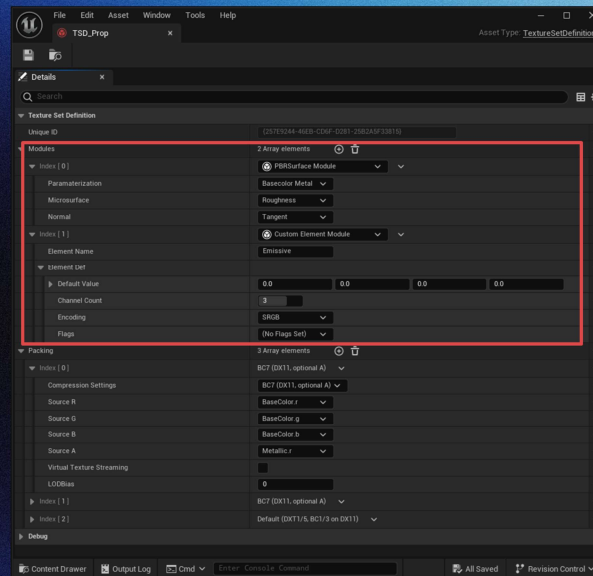
The texture set definition looks familiar.

We still have a packing definition that behaves in more or less the same way as the previous implementation. You specify an array of packed/derived textures, what elements will reside in each channel, as well as other texture settings like compression.

## UE5

### Implementation

- Definition
  - Packing Definition
    - Derived textures
    - Channel contents
  - Modules
    - Executed in order
    - Modify compilation and/or sampling
    - Attach data



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

63

I'd talked previously about how we wanted to solve the extensibility problem.

Unlike our previous implementation's hard-coded set of checkboxes, we now have this list of 'modules'.

This is our answer to extensibility.

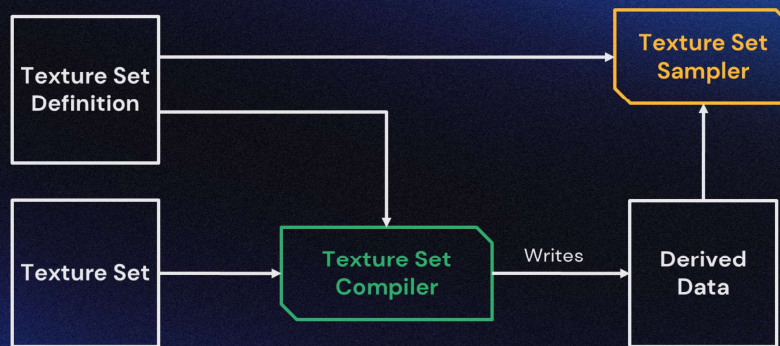
Modules are UObject derived classes that have virtual functions to customize various points in the data pipeline. Modules can author logic to determine both the process for building of the derived data on cook, as well as the sampling in the material. For predictable execution, modules are always executed in the order they appear in the definition.

They're also able to Attach data, either to a texture set or a sampler node.

## UE5

### Data Pipeline

- Compiler: Creation of derived data



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

64

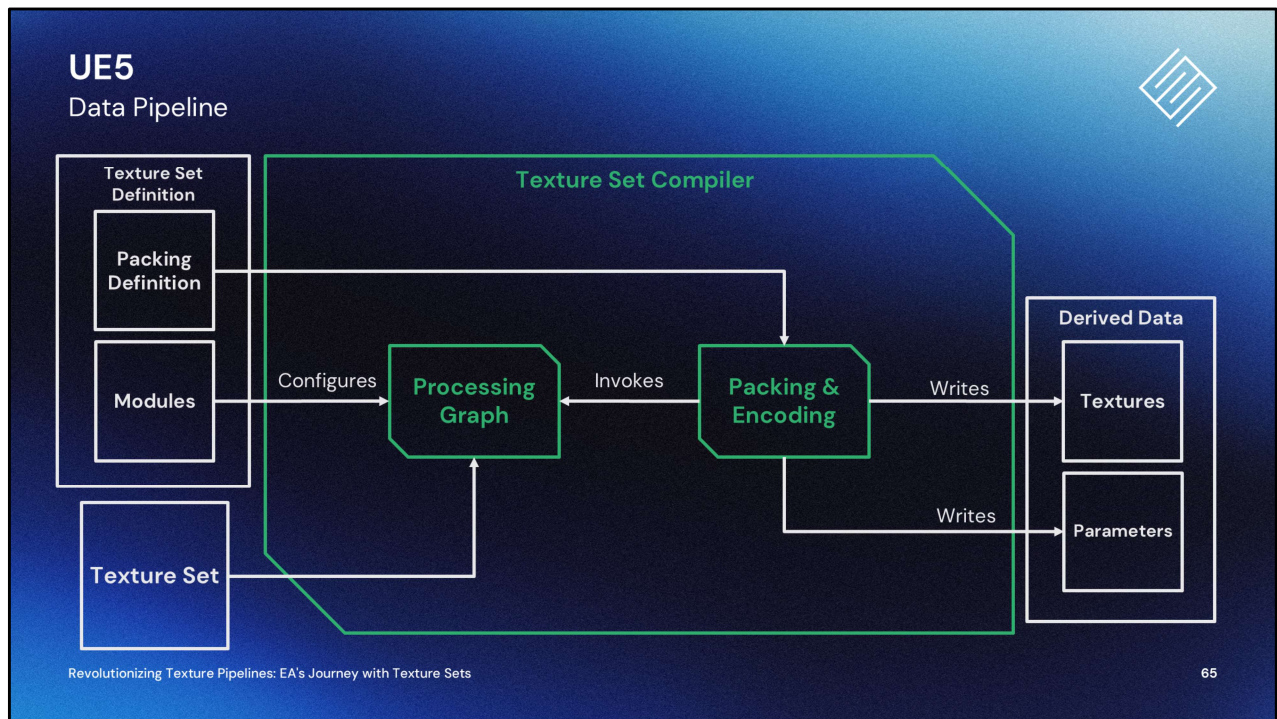
I'm going to talk a bit about what our data pipeline looks like. I've included some diagrams, but don't get caught up if you don't catch everything, they're mainly for reference - go check it out in the vault afterwards

This is an overview of our data pipeline

There's the texture set compiler, which reads both a texture set and its definition to write out derived data

Derived data is then bound to material parameters for use by the texture set sampler in the material graph





Diving into the compiler itself, it runs in two steps: Processing, and Encoding.

The Processing stage is accomplished by the processing graph,

- This is where modules are able to extend the compilation logic.
- When a processing graph is instantiated for a given texture set, each module in the definition is given a chance to customize it.

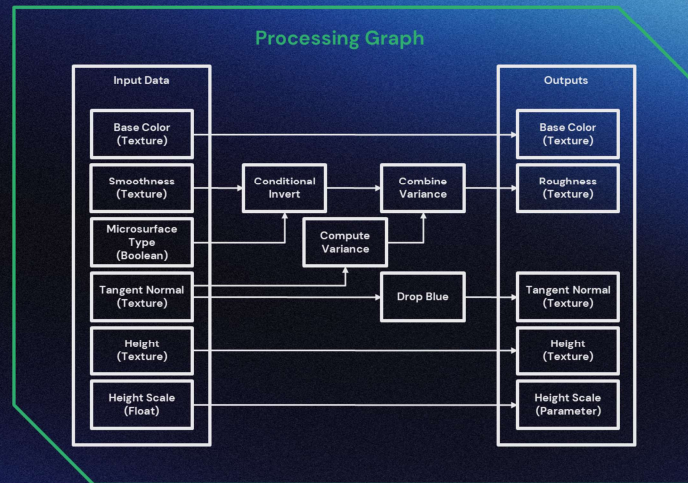
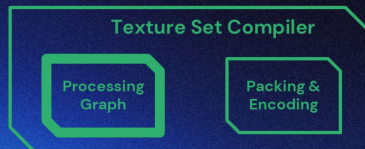
- The processing graph has two types of outputs: textures and parameters.

The encoding stage then operates on the outputs of the processing graph, and writes out the derived data.

## UE5

### Data Pipeline

- Processing Graph
  - Dependency graph
  - Inputs
    - Textures
    - Parameters
  - Outputs
    - Textures
    - Parameters



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

66

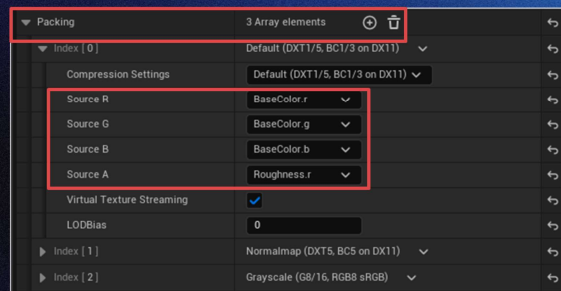
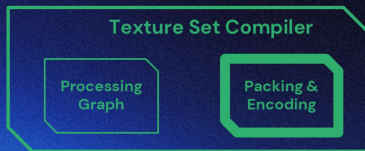
Processing graph itself is a dependency graph with inputs and outputs:

- Texture set modules are able to fully customize this graph for a given texture set. They then can add processing nodes which apply arbitrary logic, and any processing nodes can be marked as outputs, which are then available for packing and encoding into the derived data.
- Nodes can read **Input Data** from the texture set they are executing. Input data can be any UObject type: containing simple values: such as floats, integers, or booleans, to complex types: such as texture, mesh or object references.
  - When a node request input data, it will be attached to all texture sets using that definition, allowing it to be configured by users on the asset.
- Each Node can compute a recursive hash which is use to determine if a rebuild is required, as well as to cache the result with the DDC.

## UE5

### Data Pipeline

- Encoding
  - Define derived textures
  - Processed textures swizzled into channels
  - Not provided by a module



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

67

Now onto the encoding stage of the compiler.

- Encoder is hard-coded: not provided by a module. This standardizes the pipeline, abstracts data storage & compression from the modules, and it's heavily optimized to reduce compile times.
- It operates on the texture outputs of the processing graph, which we call "processed textures".
- The encoder executes the processing graph to write each processed texture into a channel of a derived texture, as defined in the packing definition.

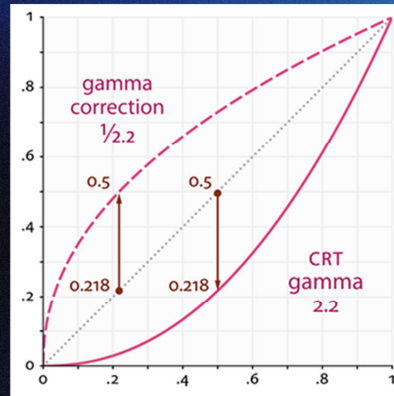
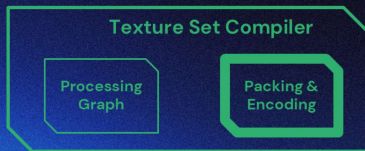
We also support a number of other features during encoding in addition to those found in standard block-compression. These are passed as flags from the processing graph inform the encoder.



## UE5

### Data Pipeline

- Encoding
  - sRGB (~gamma 2.2)



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

68

We have full sRGB Support, or more correctly, gamma 2.2 support.

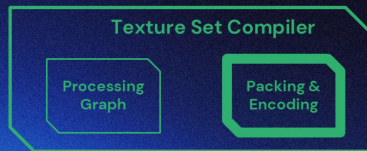
- This is only used if we're encoding to 8 bit formats. If hardware srgb is supported from the format, it will be used, otherwise conversion will be done while decoding. This allows for srgb data to be packed in BC4 and BC5 textures, as well as in the alpha channel of any other 8 bit format.

- To make use of the hardware texture sampling, gamma decoding happens after interpolation. While that's technically incorrect, we haven't run into visible issues as of yet, but if we do, we can always modify the packing of the offending definition to make use of the hardware supported formats.

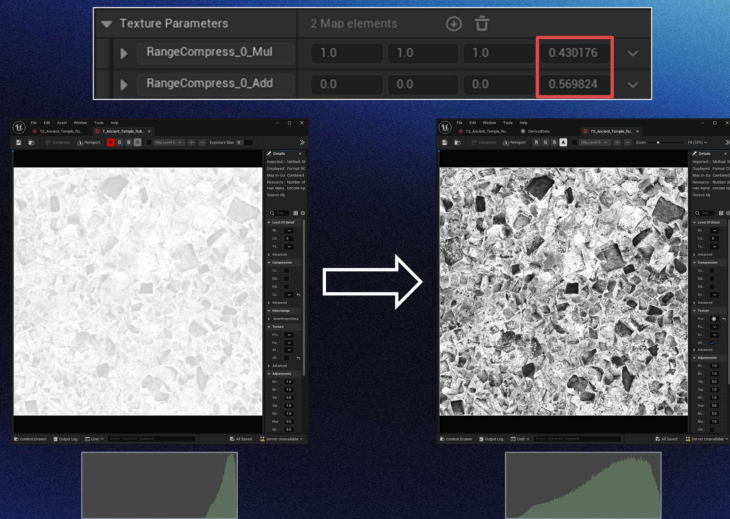
## UE5

### Data Pipeline

- Encoding
  - Range compression



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets



69

Then we have what we call "range compression"

- This is for textures that only use a limited range of values. We compute the min and max pixel value for each channel of the packed texture, and then expand the texture data to use the entire range of the texture format. When sampling, we then apply a scale and bias operation to return the samples to their original values.

The example shown here is with a roughness texture, and we show their histogram below. This works best when the raw data is imported in floating point, but as block compressed formats have limited precision for their endpoints, using the full range allows them to more accurately represent values, even if the raw data is 8 bits.

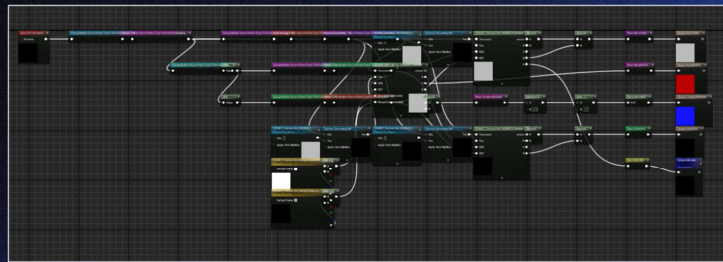
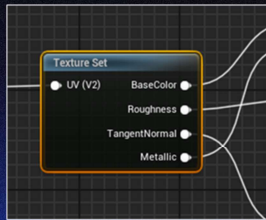
This also allows us to store values over 1 as well as negative numbers which could be useful for things like normal maps, motion vectors, or displacement.

## UE5

### Data Pipeline



- Sampling
  - Subclass of material function call expression
  - Generates material function on demand



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

70

Now on to sampling

- Texture set sampler material expression is a subclass of material function call. It generates a material function on demand based on the provided texture set definition.
- Unpacking and decoding logic that matches the packing and encoding is generated automatically.
- Similar to the processing graph, modules are able to add logic to configure the sampling graph. They're able to read the already decoded values which match the outputs from the processing graph. They can add additional logic, or simply forward the values to outputs of the material function.
- Modules are also able to request additional sub-samples, specify blend weights for the subsamples, and override the context, such as what UV or tangent space is used for that subsample. This is used to achieve techniques such as triplanar mapping, or blending between frames of a flipbook animation.
- The spaghetti on the right is an example of the generated graph. While it's not generally exposed to the user, it *is* useful for debugging, and to get an idea of what is happening inside the sampler node.



## UE5

### Standard Modules



- Standard Modules
  - Included by default
  - Generally useful
  - Standardize commonly used functionality
  - Optional

While texture sets itself no longer contains a hard-coded list elements that can be enable or disabled, we didn't want it to come out of the box with *no* functionality.

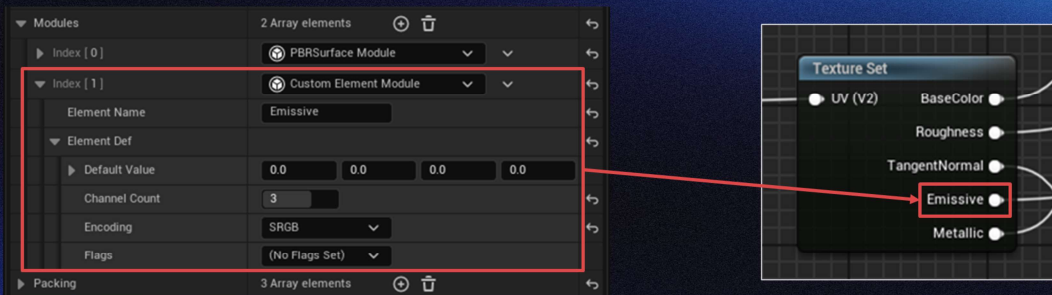
We've developed a number of what we call 'standard modules'.

- These are modules included with texture sets by default.
- They're generally useful, and not too specialized. We've tried not to overcomplicate these modules by trying to handle every possible need, but instead focused on the most commonly used functionality.
- These standard modules are optional. If you don't like them, or want to modify the implementations for your project, you can always make your own modules!
- But they also provide initial test-cases for us to validate our systems against.

# UE5

## Standard Modules

- Custom Element Module



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

72

The first, and simplest module is the custom element module. The custom element module allows a named element to be explicitly defined, without any extra processing.

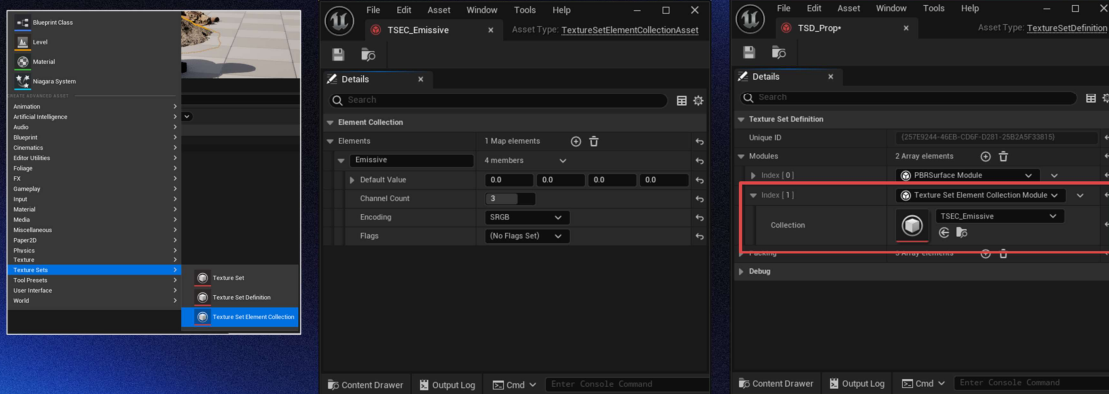
- You can specify default value, channel count, and encoding flags.
- The custom element will be added as an input on each texture set using the definition. It will flow through the pipeline without any other special processing, and will be made available as an output pin on the sampler node.

But say you have a custom element that you want to use across a few different texture set definitions?

## UE5

### Standard Modules

- Texture Set Element Collection



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

73

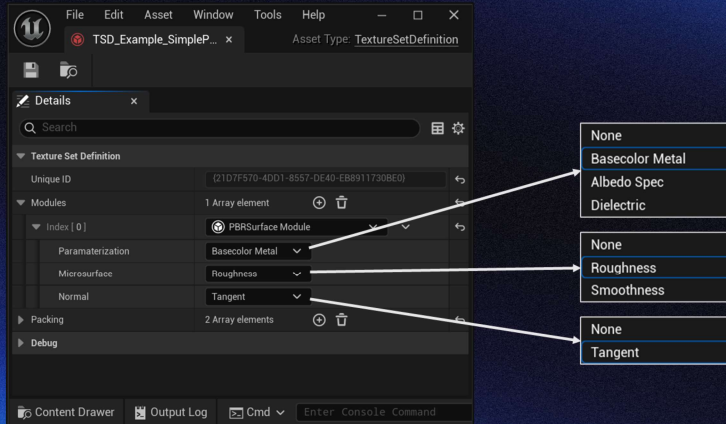
- The texture set element collection is a list of custom elements, but saved in an reusable asset.
  - The element collection *module* can reference the element collection *asset*, to include it in the definition.
- It's useful for things like custom mask types, or anything not present in the other standard modules.



# UE5

## Standard Modules

- PBR Module



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

74

We did want to further standardized the way of dealing with common PBR properties, so we created the PBR module to handle this.

On the texture set definition, it can be configured with:

- Choice of parameterization
  - Bascolor/metal
  - Albedo/spec
  - Dielectric

Microsurface can be stored in

- Roughness
- Smoothness

and has an optional tangent normal

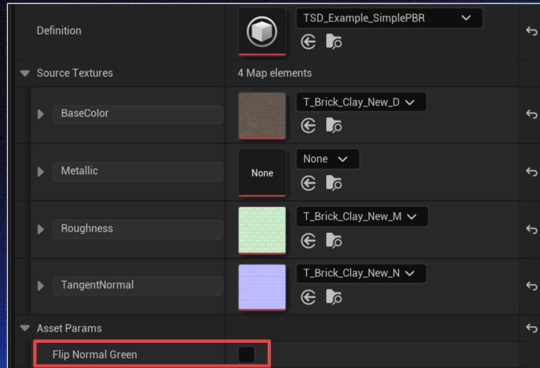
It doesn't allow explicit naming of the elements, so it keeps things consistent for all users of texture sets.

It avoids for instance the base-color sometimes being named "Color", "Diffuse", or "Albedo", and makes asset portability more straightforward.

## UE5

### Standard Modules

- PBR Module



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

75

The PBR module will add a number of input elements to the texture set based on the module's configuration.

The pbr module also adds an option to invert the normal map's green channel.

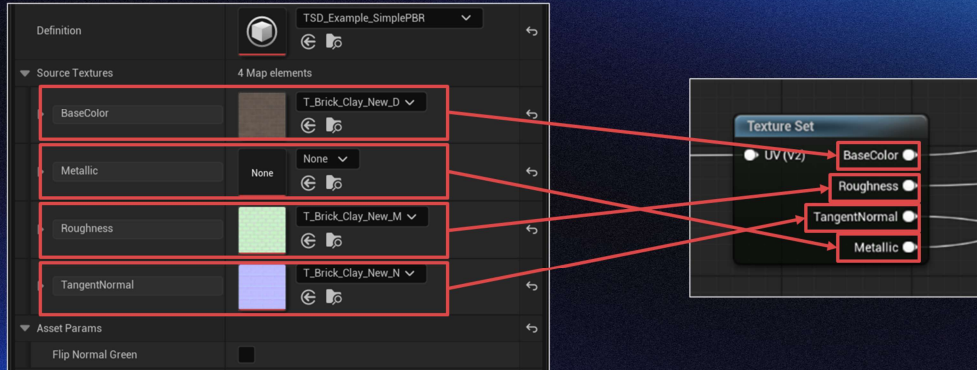
In the future additional options could be added, for instance, NDF prefiltering which is sometimes called normal to roughness. We could also support conversion between bascolor/metal and albedo/specular, or between roughness and smoothness.

When the processing graph runs, it's able to read these options and affect the built texture data.

## UE5

### Standard Modules

- PBR Module



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

76

When sampling we get a node in the graph, and it's populated with all the outputs we expect. Since we know what each element is, the pbr module is able to automatically handle transforming the tangent normal from the 0 to 1, to the -1 to 1 space. It's also been stored as a 2 channels, so it handles reconstructing the Z value.

It's common for people new to material graphs to get tripped up on transforming normals to world space...



## UE5

### Standard Modules

- PBR Module



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

77

So to help make things more intuitive, the PBR module attaches some parameters that can be configured on the sampler node. One parameter is the space of the normal output, which by default is in tangent space.

If the material artist wants to get the normal in world space, they can change the option to world space and the transform is handled internally, now outputting the world space normal.

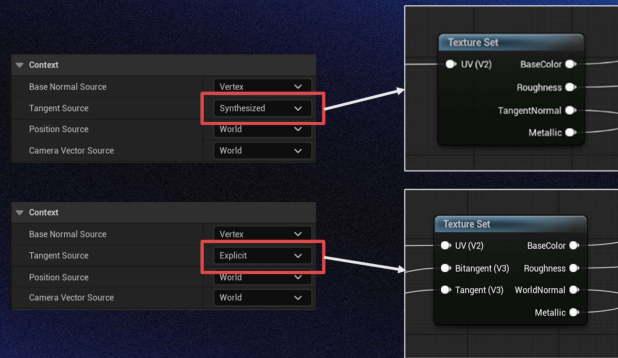
But wait, you may or may not be saying, what tangent space is used for that transformation? Well, by default, the tangent space that's stored in your mesh's vertex data is used. But this only works for the first UV set, and doesn't work for secondary UV sets, or if you use some other projection mapping.

Well, we have that covered too...

## UE5

### Standard Modules

- PBR Module



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

78

Every sampler node has a number of options to specify the 'context'. In this case, we can change the tangent source to be synthesized, which will instead compute the tangent space based on screen-space derivatives for whatever UV set is passed in. This allows us to support arbitrary UV spaces and projections.

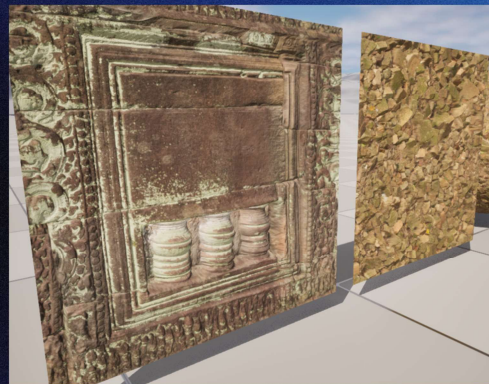
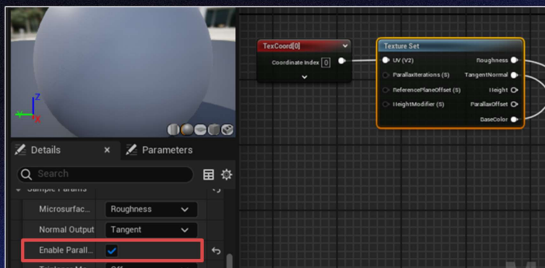
- You're also able to change the tangent source to "explicit" to feed in your own tangent space and skip the cost of the calculation altogether.

- This concept of 'context' is supported for a few other values, namely the base normal, position, and camera vector. This flexibility helps us maintain full support for different material domains, such as post-process or volume materials. It also ensures that various types of modules remain in-sync, and avoid redundant calculations.

## UE5

### Standard Modules

- Height Module



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

79

Moving on, we also created the height module, which provides a heightmap to the texture set.

It allows parallax occlusion mapping to be enabled on the sampler node

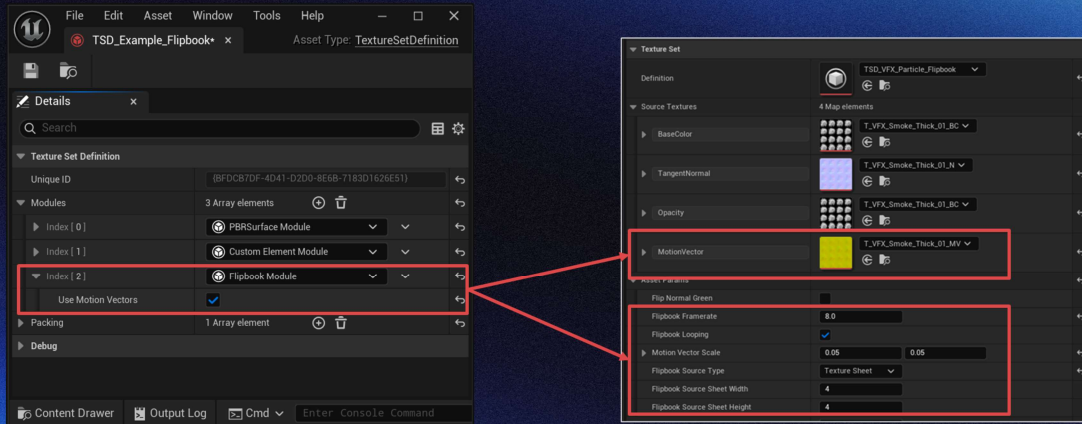
This is also another good use-case of range compression. Artists can safely use a limited range of the heightmap without worrying about wasting samples, as we ensure we only step through height ranges that have valid data.



# UE5

## Standard Modules

- Flipbook



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

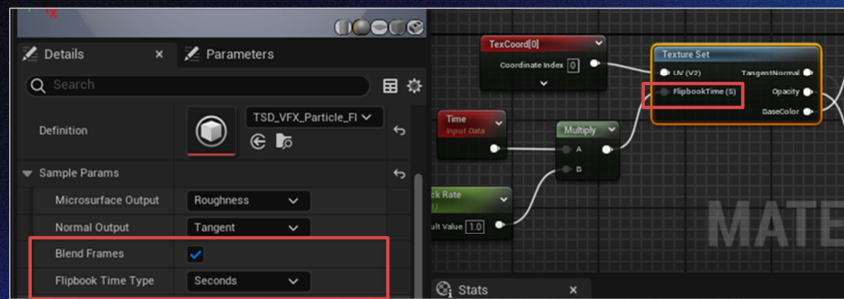
80

- The flipbook module turns each texture set into a flipbook animation, and
- It uses texture arrays internally, but can read in texture sheets.
  - It attaches parameters to each texture set where things like texture sheet dimensions, default flipbook framerate, or looping of the flipbook can be specified.
  - Motion vectors can also be enabled, which are used to warp one frame into the next, reducing the minimum acceptable framerate, and allowing for smoother animations.

## UE5

### Standard Modules

- Flipbook



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

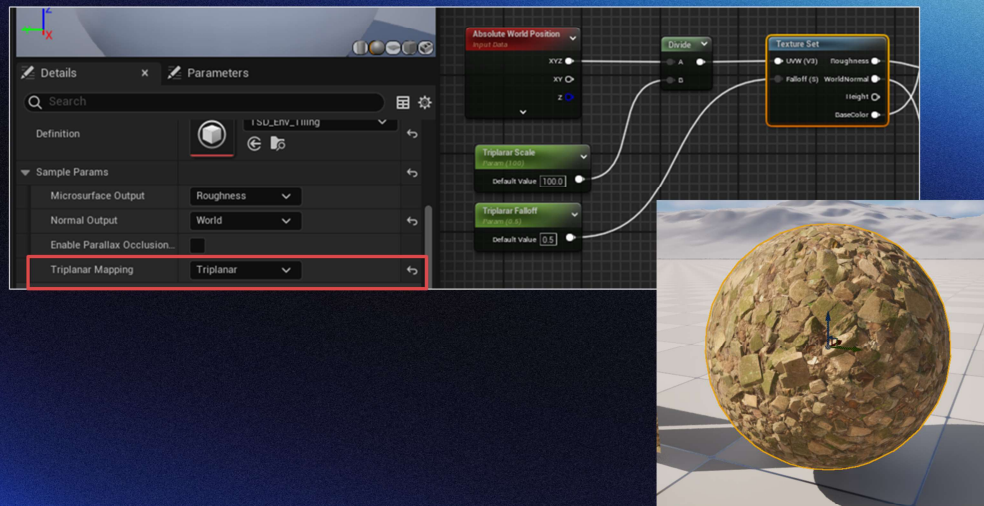
81

- On the sampler node, an additional input pin for 'flipbook time' is added, which is used to drive the playback of the flipbook animation.
- An option is added to the node's properties to specify how time is considered, be it by frame, normalized between 0 and 1, or in seconds, using the default frame rate specified on the texture set.
- Another option for blending between frames can be enabled with a simple checkbox.

## UE5

### Standard Modules

- Triplanar



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

82

Finally, we have the triplanar mapping module.

It's added to the texture set definition like any other module, but this module doesn't add any data to texture sets, or modify the processing in any way. Instead, it enables the option of triplanar mapping on the sampler node.

This requests multiple sub-samples within the sample node, computes appropriate blend weights, and then overrides the UV and tangent spaces for each of the 3 subsamples.

Since it computes appropriate tangent spaces, this ensures that other modules that rely on them, such as the PBR module for normal maps, still function correctly.



UE5

Standard Modules



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

83

It's worth noting, that these modules are all able to be used in combination. Here, we have an example of parallax occlusion being used in alongside triplanar mapping.

## UE5

### Challenges



- Plugin limitations
- Asset Pipeline
- Context permutations
  - Debug/Release
  - Editor/Game/Play in Editor / Commandlet(s)
  - With/Without cooked data
  - With/Without editor data

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

84

Now, we did have some challenges with the engine, while working on the tech. Much of this is probably old news to developers familiar with the engine.

- We were limited in what we could do from a plugin. Development would have been much smoother as an engine feature, or with more divergences, but that would have posed issues for adoption and maintenance costs.

- UE architecture means that the engine, the editor, and the data pipeline are all the same application.

- This requires data to always be in a runnable state, even if unbuilt.

- We needed to handle async building of the data, while the user could also be editing the underlying objects.

- And finally, we need to be able to hot-swap the built data into the game when complete.

- Unfortunately the engine provides only a loose framework for this, and asset type is required to manage it's own tasks/threads/memory usage while compiling asynchronously.

Also ran into issues with asset dependencies. Pattern of one asset depending heavily on another to cook are sparse.

- Resulted in us needing to manually fire events and callbacks to keep everything in sync.

- The number of context permutations is also large: we have Debug/Release + Editor/Game/Play in Editor / Commandlet(s) + With/Without cooked data + With/Without

editor data

We'd run into cases such as a crash that occurred only when data was not found in the DDC, building HLODs, in a commandlet, on a build machine.

We can fix issues we come across them, and make sure it works for our projects, but it's been hard to be confident it will work across all of the many contexts and use-cases.





## Chapter 5: Where are we Now?

Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

85

Chapter 5: Where are we Now?

## Where are we now?

### Things to do

- Continuing to stabilize
- Import Pipeline
  - Support .psd, .exr, .sbsar, loose textures, etc.
- Encoding
  - Extensibility
  - Beyond texture packing
    - Neural texture compression? <https://github.com/NVIDIA-RTX/RTXNTC>

We want to continue to stabilize, as mentioned before, there's sure to be bugs and crashes we haven't yet discovered.

We'd love to implement the import pipeline, to support various input formats, such as the PSD and EXR of our previous implementation, as well as other formats such as substance archives, and even individual (loose) textures.

There's also a great opportunity to extend the encoding mechanism beyond texture packing. This framework seems like an ideal match for neural texture compression.

## Where are we now?

### Things to do

- More modules!
  - Texture bombing [Glanville]
  - Relaxed cone step mapping [Policarpo]
  - Flipbooks with motion vectors computed during processing
  - Map baker
    - AO / Cavity / Normal maps
  - Mip flooding [Feeley 18]
  - ... and more!



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

87

Would also love to build out more modules, as there are tons of possibilities and we've got no shortage of ideas!

To touch on a few:

- Texture bombing - which would be a sampler-only module like triplanar mapping, but used to reduce the appearance of texture tiling
- Relaxed cone step mapping – notoriously hard to use in production due to its need to pre-process the heightmap, but should now be much easier!
- Since we can create new textures based on existing textures, flipbooks could have their motion vectors automatically computed during the pipeline, instead of done externally in a DCC
- If we're computing new textures, then we could implement a map baker for AO, Cavity, or Normal maps, which would be computed based on a texture set referencing a source mesh.
- We could include mip flooding to fill all data in transparent areas with a constant value for more efficient compression.

And if we're able to reference a mesh at build time, we could do the same same for pixels outside the UV shells on texture sets without an opacity map.



## Where are we now?

The landscape

- One project in production using Texture Sets
- Other teams at EA interested in adoption
- (New) Frostbite version in the works



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

88

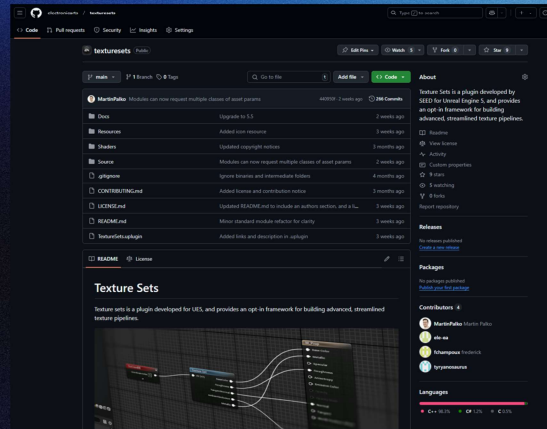
The landscape within ea look a bit like this:

- We've got one project in development, using texture sets for UE5.
- Multiple other teams have expressed interest
- And there's a new frostbite version in the work

## Where are we now?

The landscape

- Open source!
  - Plugin:  
[github.com/electronicarts/texturesets](https://github.com/electronicarts/texturesets)
  - Engine changes:  
[github.com/MartinPalko/UnrealEngine](https://github.com/MartinPalko/UnrealEngine)  
(accessible to UE licensees)



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

89

We've also chosen to open source the Unreal plugin on github, so you can check out the code for yourself!

The plugin is available on EA's github page, and there's a second repository available to UE licensees which contains the engine changes required to get up and running.

- You should consider the code up on github *\*experimental\**. There are some rough edges, and while we do have a project using it in production, they have a number of engineers who are familiar with the tech to help them when things go wrong!

## Takeaways

If you remember nothing else



1. Texture sets are great!
2. Sometimes we don't do things not because it's not possible, but because it's not practical
3. Computer science principles apply to content too
  - Power of decoupling intent and implementation
  - Don't repeat yourself

So if you remember nothing else from this presentation, remember:

1. Texture sets are great! We've open-sourced them so you can use them, so no excuses! Textures have been stuck in a rut for a while, and we need to move on!

2. sometimes we don't do things not because it's not possible, but because it's not practical

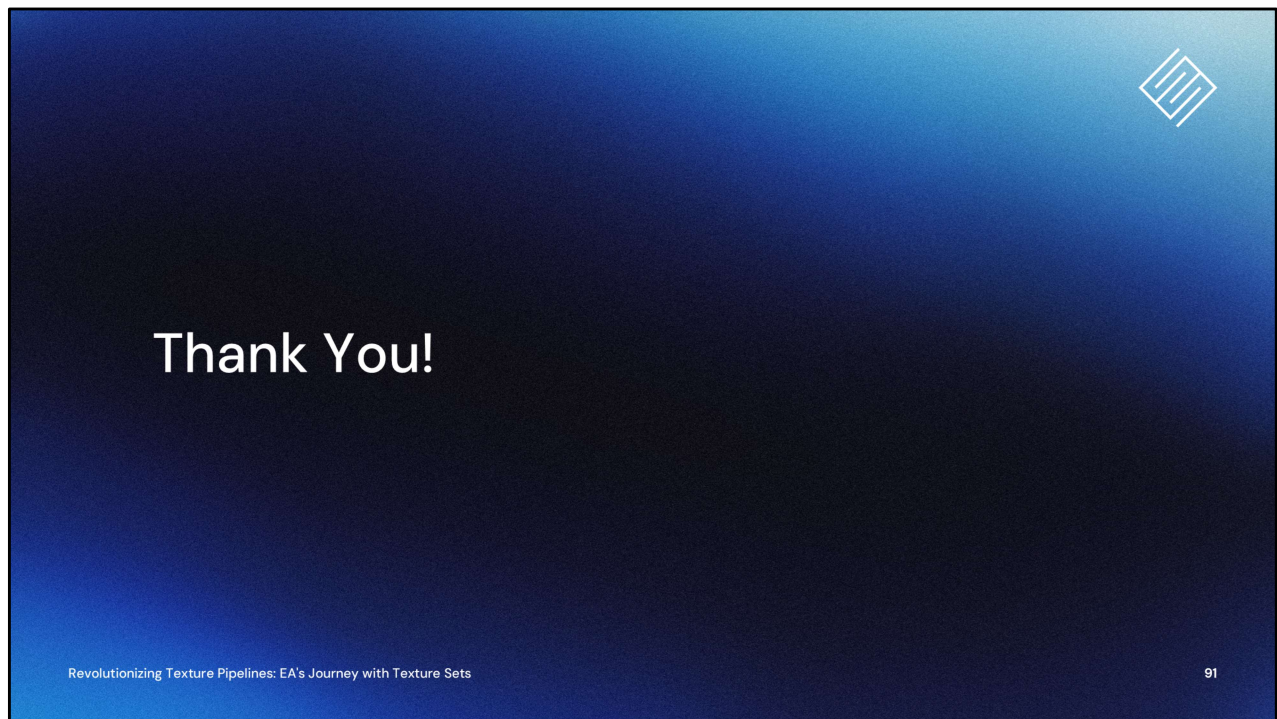
- These need engineering too! No technique that texture sets can do, was impossible to do before. But many of them were difficult or impractical

3. Computer science principles apply to content too!

- Power of decoupling intent and implementation. Draw a line between what it is/how it's implemented. This is the same mentality between the concept of APIs and interfaces in computer science. It allows tech to evolve and be optimized without content needing to be re-authored

- Don't repeat yourself, also known as DRY principle. It applies to data too! Too often, properties and settings are duplicated hundreds or thousands of times across a project. Good systems will replace duplications with references.





Thank you!

**We're changing  
the game.  
Join us!**

Rendering, Computer Vision, AI/ML,  
Physics, Animation.

Games, Engine, Central Tech, Applied Research


**Electronic Arts**




[ea.com/careers](https://ea.com/careers)

EA and SEED are always looking for talent, and hiring across a range of disciplines: from game teams to central tech, to applied research.

If you want to check out current positions, you can either scan this QR code, or go the link shown under it.



# Questions?



Revolutionizing Texture Pipelines: EA's Journey with Texture Sets

[ea.com/careers](https://ea.com/careers) 93

Thank you again!  
Time for questions?



## References



- [Glanville] GPU Gems: Chapter 20. Texture Bombing
  - <https://developer.nvidia.com/gpugems/gpugems/part-iii-materials/chapter-20-texture-bombing>
- [Policarpo] GPU Gems 3: Chapter 18. Relaxed Cone Stepping for Relief Mapping
  - <https://developer.nvidia.com/gpugems/gpugems3/part-iii-rendering/chapter-18-relaxed-cone-stepping-relief-mapping>
- [Feeley 18] Interactive Wind and Vegetation in 'God of War'
  - [https://www.youtube.com/watch?v=MKX45\\_rIWQA&t=2954s](https://www.youtube.com/watch?v=MKX45_rIWQA&t=2954s)

So if you remember nothing else from this presentation, remember:

1. sometimes we don't do things not because it's not possible, but because it's not practical
  - These need engineering too! No technique that texture sets can do, was impossible to do before. But many of them were difficult or impractical
2. Power of decoupling intent and implementation
  - Draw a line between what it is/how it's implemented. This is the same mentality between the concept of APIs and interfaces in computer science. It allows tech to evolve and be optimized without content needing to be re-authored
  - Don't repeat yourself (DRY principle) applies to data too. Too often, properties and settings are duplicated hundreds or thousands of times across a project. Good systems will replace duplications with references.