Introduction

- Dragon age: The Veilguard - the latest installment in Dragon Age series.
- Developed on Frostbite Engine.
- Launched on PC, PS5 and XBSX/S.
- Small rendering team at BioWare.
- ~5 rendering engineers.
- Supported by Frostbite team.

Agenda

- Motivation
- Appearance Editor
    - Creating Appearances in Editor
    - Loading Appearances in Game
    - Appearance System
- Character Creator
    - Character Creator Architecture
    - Customizations
- Strand hair
    - Lighting Strand Hair
    - Strand Hair Performance
- Infill Characters
- Dynamic Shader Variation System
- Gear System
- Takeaways

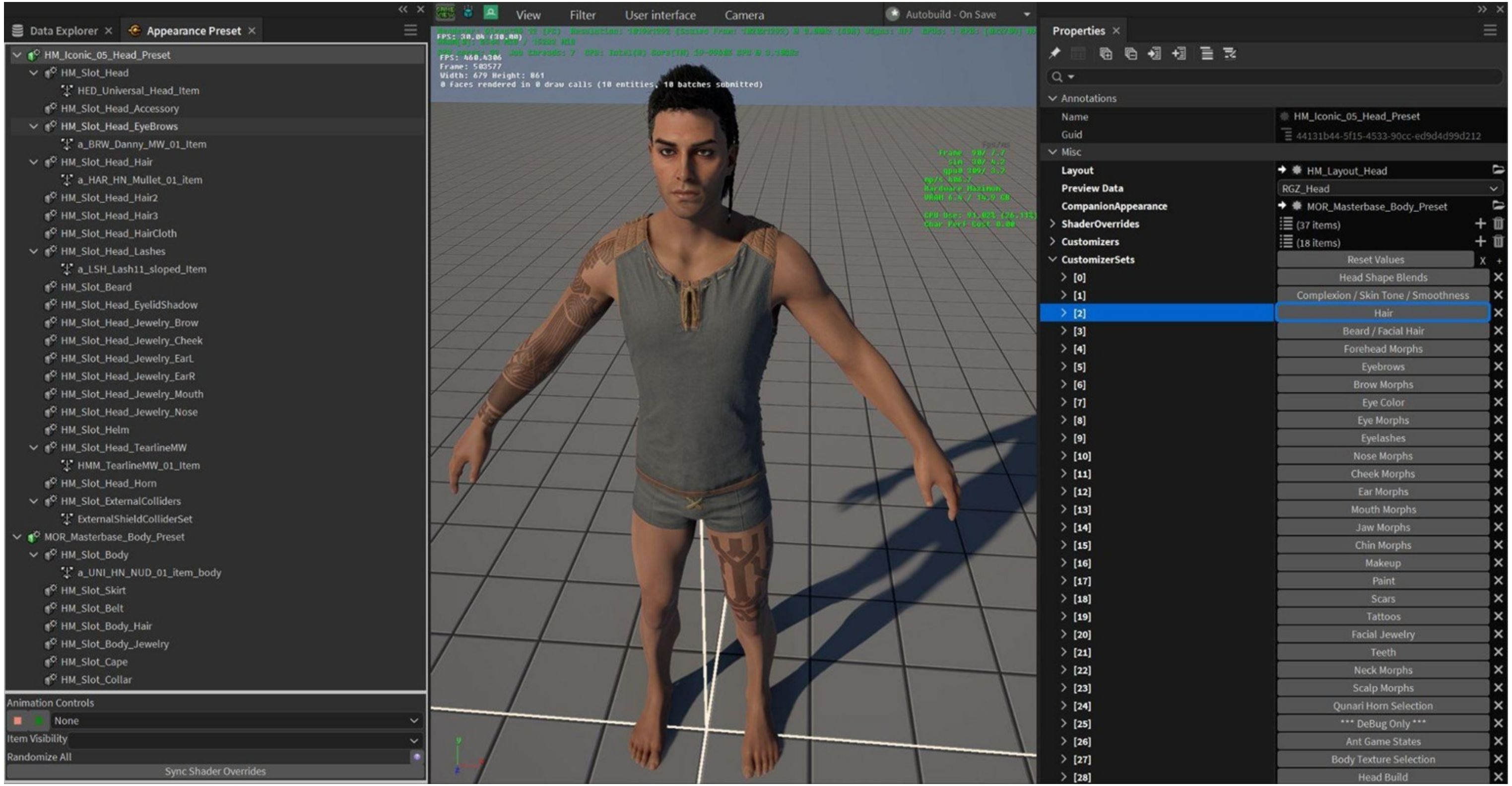
Character Examples

Electronic Arts

Motivation

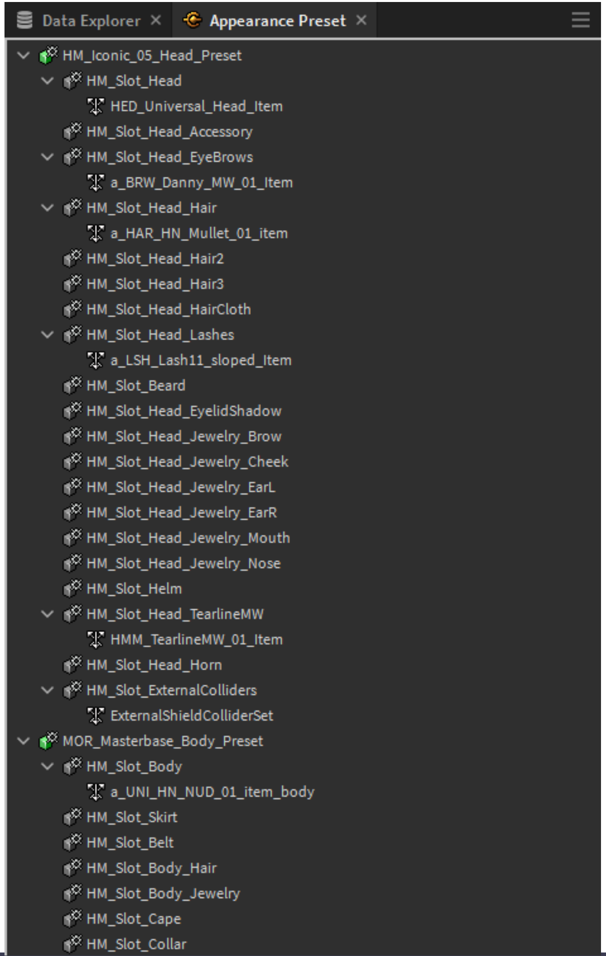
- Provide artists enough options to create a variety of characters.
- What you see in editor is what you get in game.
- Common rendering subsystem for both editor and in-game character creator.
- Dev tools were needed before in-game character creator.
- Fast iteration.
- Push the boundary on number of customization options in-game.
- Rendering performance estimates could be in done in editor.

Appearance Editor



Creating Appearance in Editor

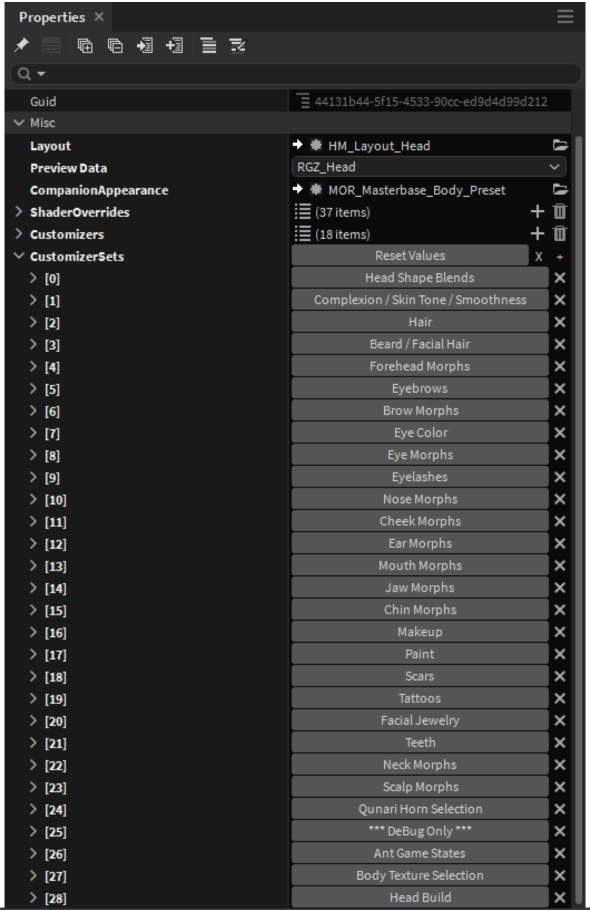
- Appearances are divided into 2 parts - head and body.
  - Allows variety in character groups.
  - Head can be a blend of upto 3 head meshes.
- Each part has slots which are authored by artists.
- One appearance item per slot at a time.



Creating Appearance in Editor

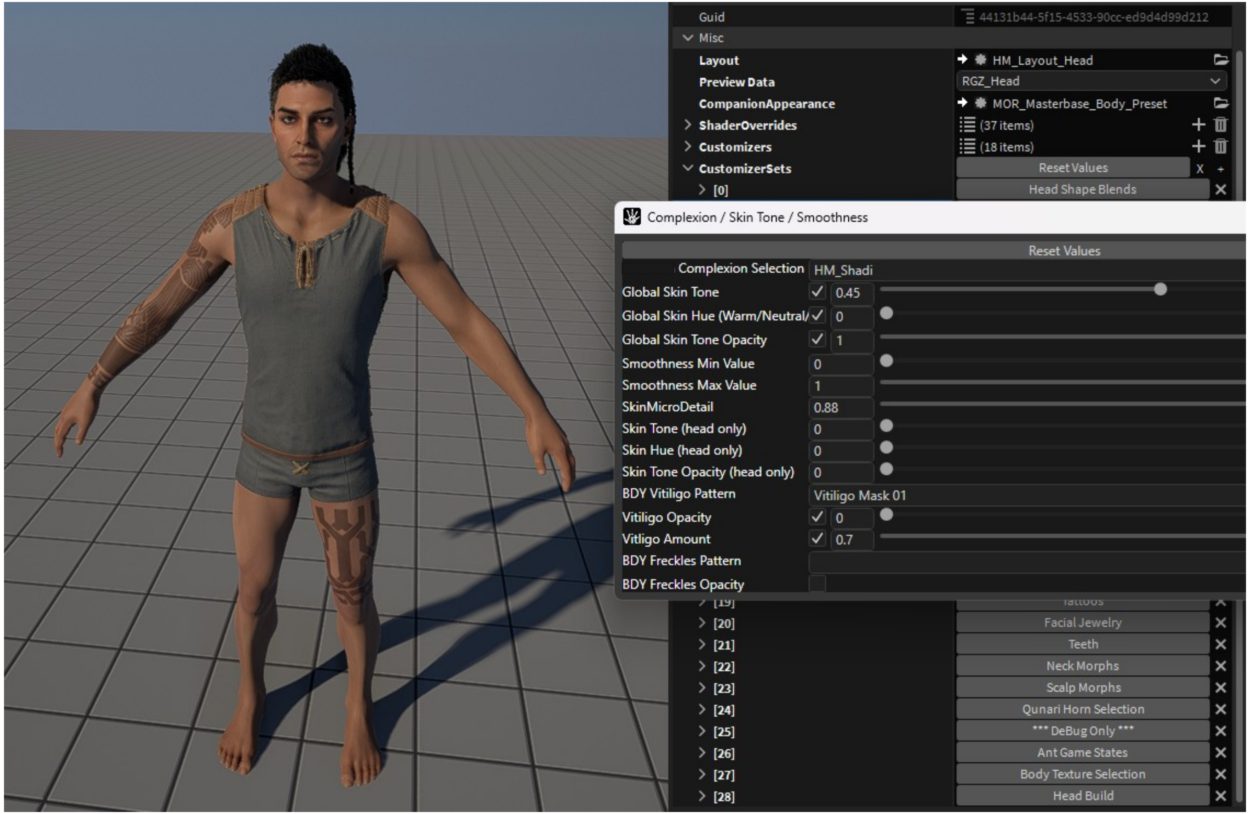
This panel allowed following customizations:

- Assign appearance items to slots
- Choose custom colors for hair and eyes
- Morph face and body features
- Apply texture sets for makeup, complexions, tattoos etc
- Choose lineage and gender
- Choose head blends
- Choose skin tone
- Choose outfits and jewelry

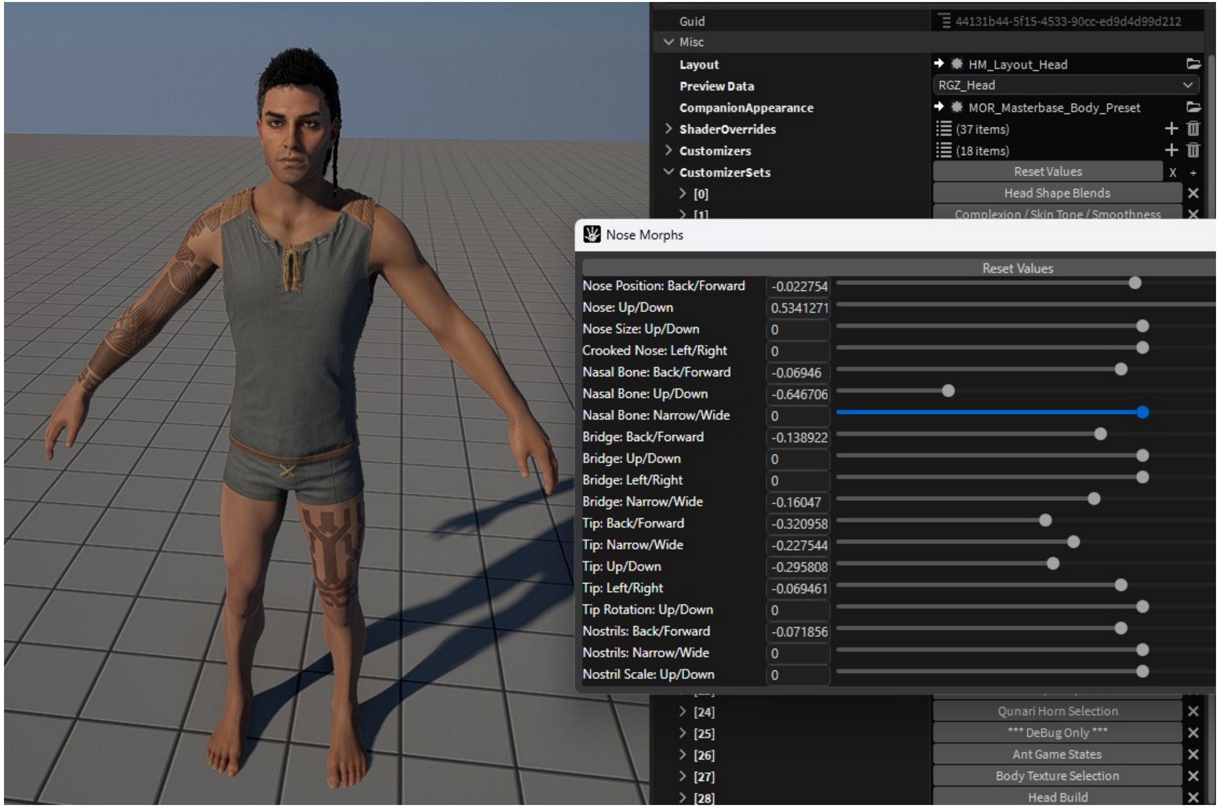


Electronic Arts

Creating Appearance in Editor



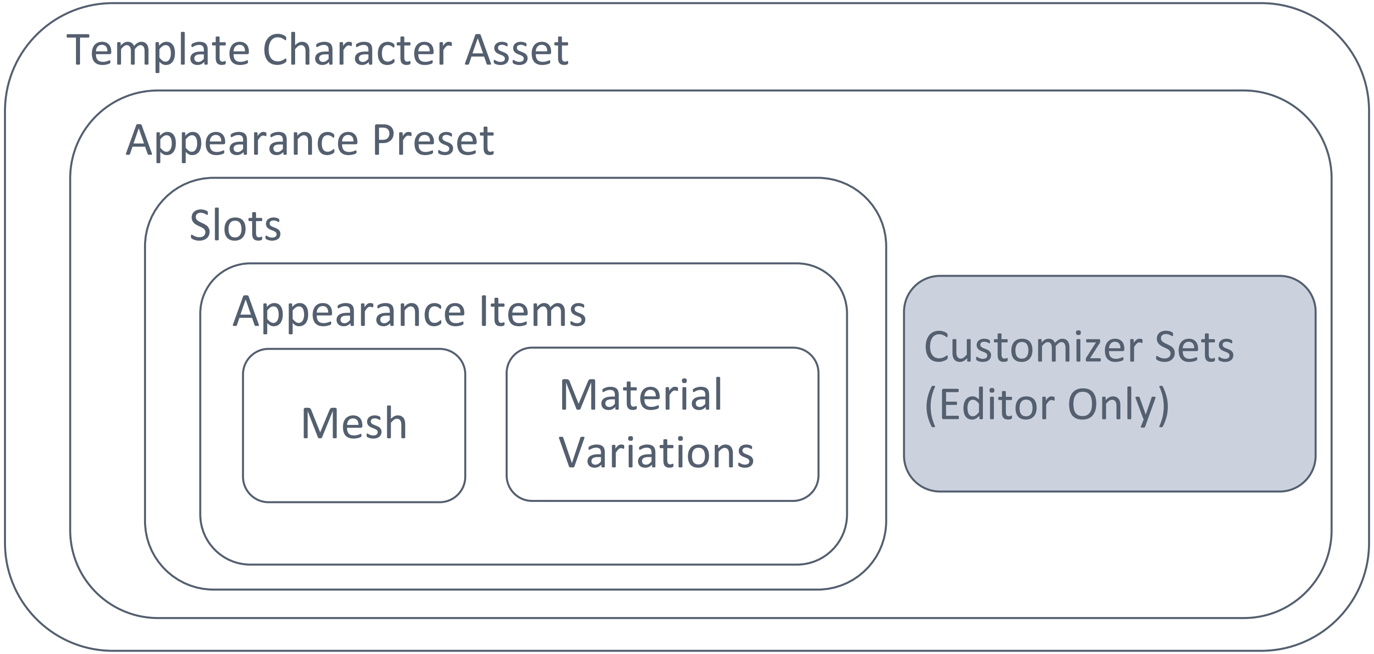
Creating Appearance in Editor



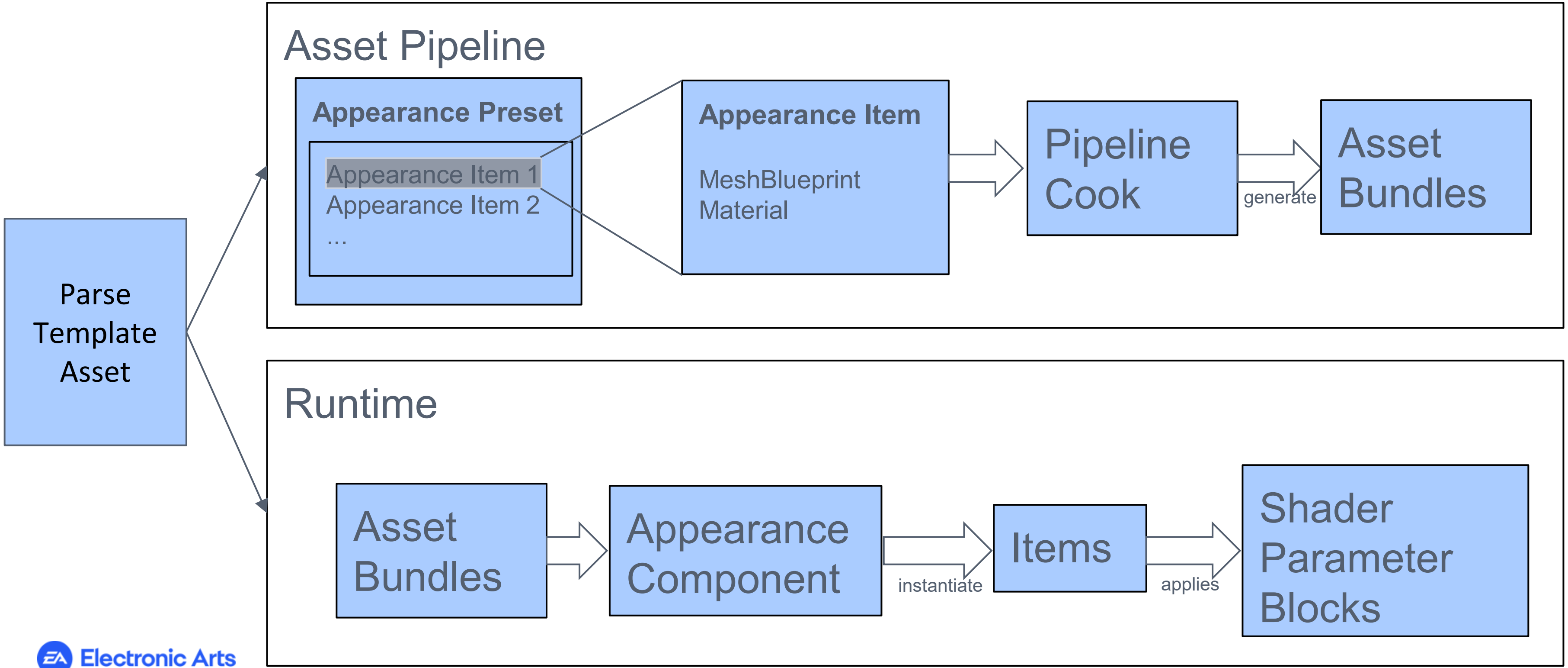
Loading Appearances in Game

Electronic Arts

Loading Appearance in Game



Appearance System



Appearance System

Appearance system is used for all character rendering. Some examples are:

- **Character Creator** : For main player character customizations
- **Infill Characters** : Randomized seed based customizations for crowd
- **Dynamic Shader Variation System** : Combat effects and injured variations
- **Gear System** : Customizable outfits that can be applied to player and followers
- **Other NPCs** : Non customizable artist authored characters and enemies

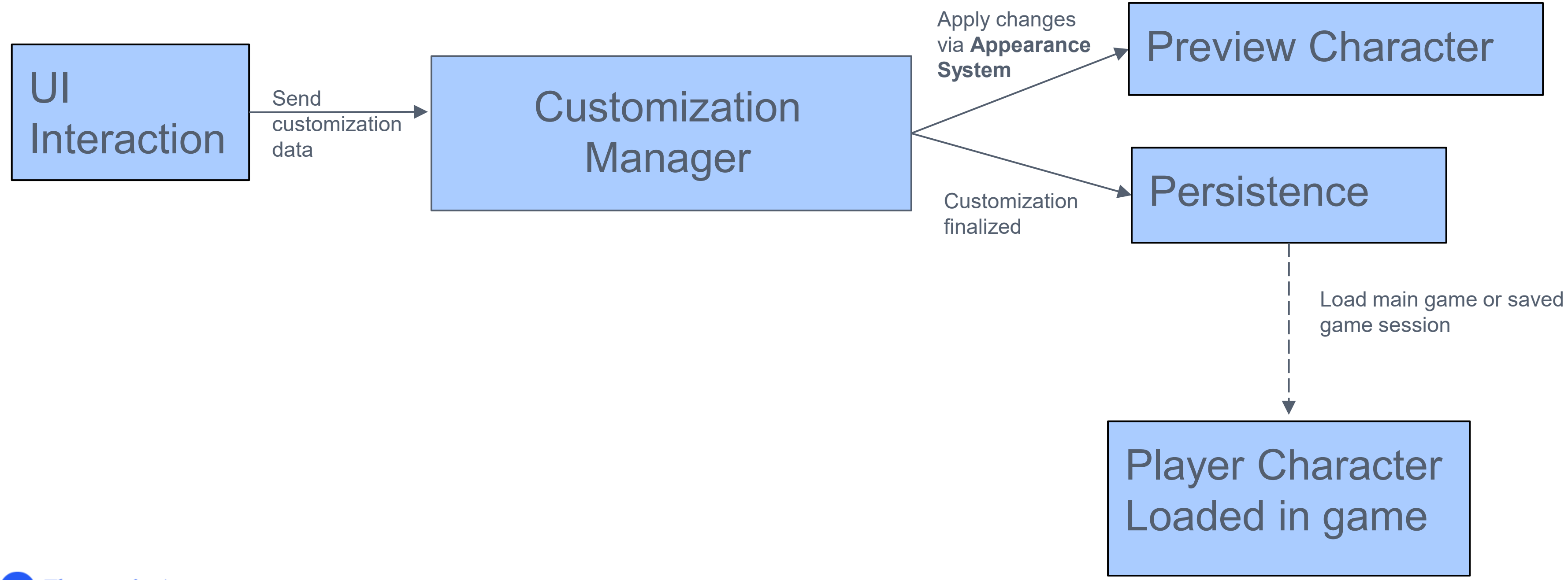
Types of NPCs in game:

- Full NPCs
- Light NPCs
- Swarm NPCs

Character Creator

Electronic Arts

Character Creator Architecture



Customizations

Customizations are divided into following categories:

- Shader Params
- Morph Features
- Head Blends
- Texture Sets
- Appearance Items
- Animation Game States

Shader Parameters

- Change applied to materials via shader parameters.
- Examples: skin tone, hair color, eye color, makeup etc.
- The shader graphs are set in such way that it allows all values in different character shaders are updated for one parameter name.
    - Allows matching face and body skin tone without separate process.
- All shader changes are applied as one shader parameter block to minimize performance hit.
- Shader parameter blocks can be stacked.

Morph Features

- Changes to facial features and body builds.
- The weights dictated how the joints in face and body rig can be changed to produce desired features.
- Weight ranges can be mapped from absolute values on rig to -1 to 1 in game.

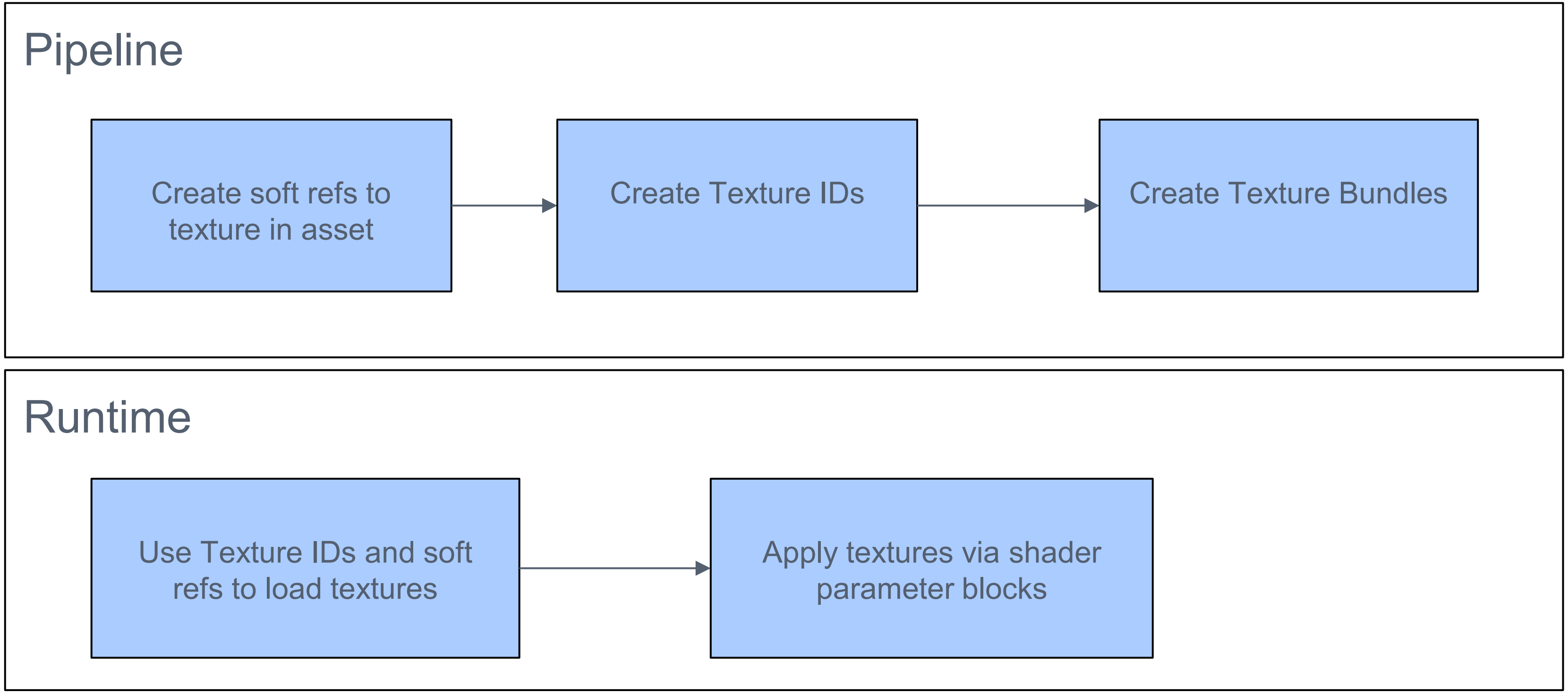
Head Blends

- 3 preset head blend shapes are interpolated between each other.
- Each race has its own collection of head blends created by artists.
- Only limited to heads, due to heavy computation.
- Memory consumption can increase based on number of head assets.
    - Solution: only keep the assets bundles that are actually used in the character and avoid duplication of asset bundles.

Texture Sets

- Used for complexion sets, makeup, tattoos, paint and scars in face and body.
- Higher memory consumption leading to lower resolution streaming textures.
    - Solution: unused bundles are not loaded into the memory.
    - Use of soft refs in the assets instead of hard references to texture assets.

Texture Bundles

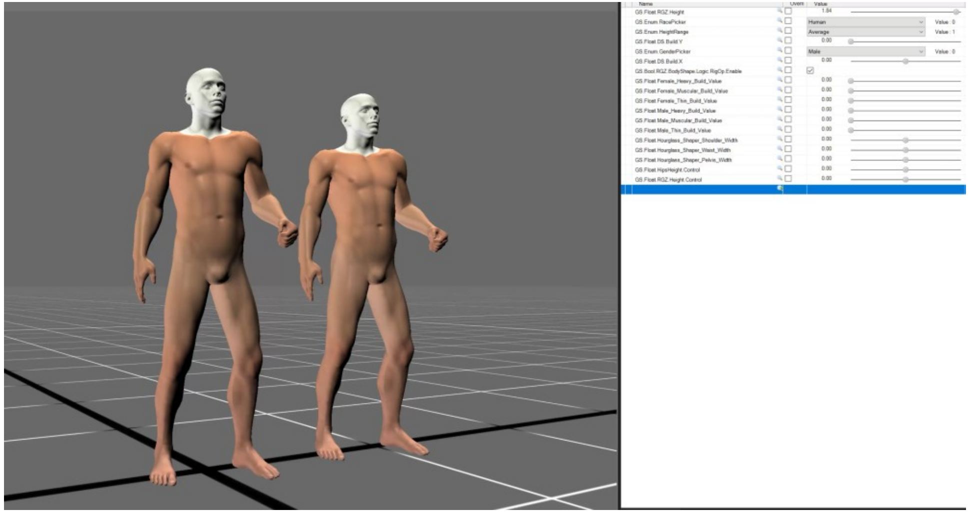


Appearance Items

- Head presets, body presets, hair, eyebrows, eyelashes, qunari horns and gear are all appearance items.
- Appearance items automatically applies all the shader overrides as setup in data.
- Async loading of items helped in achieving better performance.
- Item bundles had to be unloaded after each category was done to keep memory within budget.

Animation Game State

- Allow rig to interpolate from a male to female base and between different lineages.
- Used to control various parameters like height, build etc.
- Applies on top of a base model.
- Also manipulates the mesh skinning based on rig values.



Animation Game State

Pros:

- Easy controls for the rig exposed to artist and player.
- Easy transition between lineages and builds.
- Same outfits worked for different lineages.
- Same animation set works on all lineages.
- Rig transforms are applied once every frame in editor, but only once on reset at runtime.

Cons:

- Took a lot of time to setup correctly on the tech anim side.
- Animation set for base human model sometimes did not look correct in case of extreme values in other lineages.
- Non uniform scaling was required which broke cloth retargeting.

Strand Hair

- One of the most impressive features in Dragon Age.
- Difficult and complex hairstyles.
- Worked extensively with Frostbite engineers.
- Optimizations for performance and memory usage.
- Industry leading technology.

Lighting Strand Hair

Problems:

- Varied fantasy visual environments.
- Lighter hair color tone was prone to unexpected changes.
- The hair lighting model has no diffuse component, it's purely specular

Solutions:

- Fake character lights.
- At least one shadow casting light had to be present.
- LRVs and sky sampling for correct color.

For more details check out: https://www.ea.com/technology/news/strand-hair-dragon-age-the-veilguard

Lighting Strand Hair

A new technique for blending hair with transparent VFX and participating media.

- Split the hair into two distinct passes, first opaque, and then transparent.
- Add alpha cutoff to the render pass that composites the hair with the world and first renders the hair that is above the cutoff (opaque), and then below the cutoff (transparent).
- Before these split passes are rendered, we render the depth of the transparent part of the strand hair which is used as a spatial barrier between transparent pixels that are "under" and "on top" of the strand hair.

Lighting Strand Hair

For more details check out: https://www.ea.com/technology/news/strand-hair-dragon-age-the-veilguard

Electronic Arts

Lighting Strand Hair

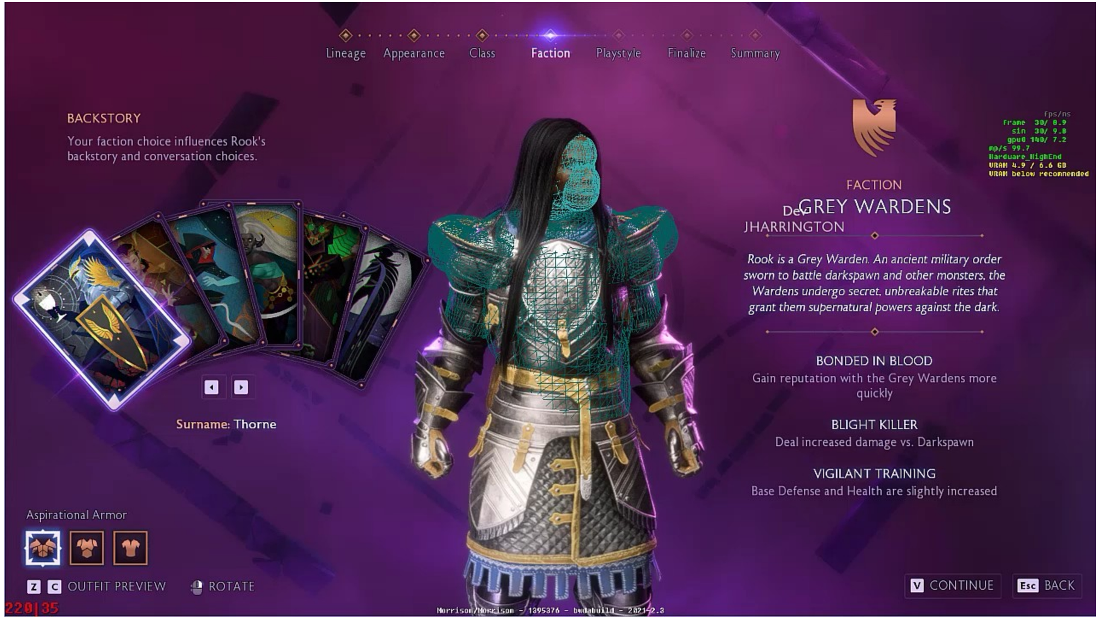
- High quality shadow maps are required in order to have good coverage of the tens of thousands of thin hair strands in the shadow map texture
- Hero shadows are rendered for every Strand Hair object and light designated as important to the shot.
- These hero shadows are generated at run time, using a tightly fitting light frustum that is adjusted to each hair's bounding box.
- As an optimization, check regular shadow maps first and if no hit, then sample strand hair hero shadows.

Electronic Arts

For more details check out: https://www.ea.com/technology/news/strand-hair-dragon-age-the-veilguard

Collider Sets

- Collider sets are used for preventing hair clipping with character outfits.
- Since the main player model proportions can be customized, the colliders also needed to fit the customization
- Colliders were moved based on skeletal joint transforms



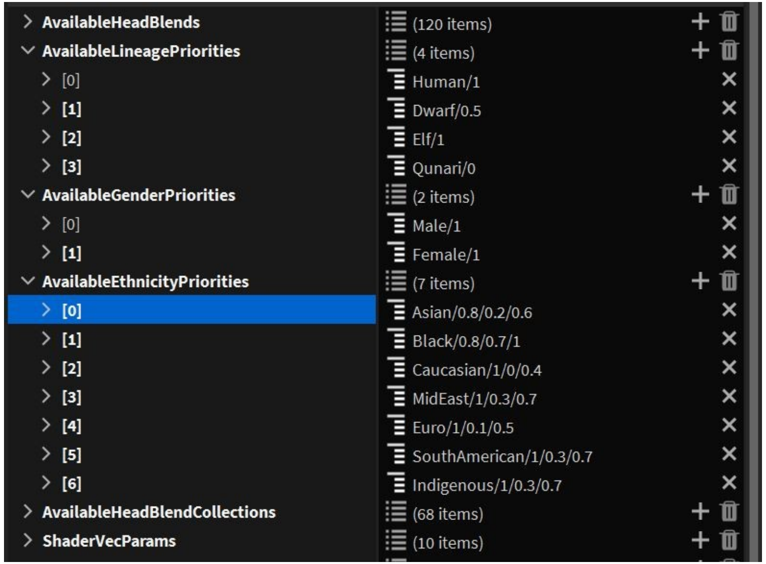
Electronic Arts

Strand Hair Performance

- **Rendering budget:**
  - 8 strand hair assets at once
  - 6.5ms for 33.3ms frametime and 3ms for 16.6ms frame time
  - Hair sim on GPU compute ~2 ms
    - Occasional spike when loading or teleporting characters.
- **Optimizations & Constraints:**
  - Only on PC high and ultra settings, PS5 and XBSX.
  - Only for head hair and beards. Eyelashes and eyebrows used card hair.
  - Only for named NPCs besides the player character.
  - Bundling the strand and card hair separately saved a lot of memory issues, especially on consoles.
  - Hair resolution control adjusts the hair resolution depending on upsampler and DRS settings.
  - Disable hair sim when off screen or far away from camera using the sim lods.
  - Decimation of render strands based on camera distance.

Infill Characters

- Values are randomized based on game lore constraints.
- These are specified in a infill character database and priority value is assigned based on required gender/lineage/ethnicity for different regions game.
- The database is parsed through the during pipeline stage and appropriate key value pairs are made from ID and asset bundles.
- At runtime, based on priority values, appropriate asset bundles are loaded and characters are rendered.
- The appearance system was reused to allow for variations in the infill characters but did not customize at runtime.



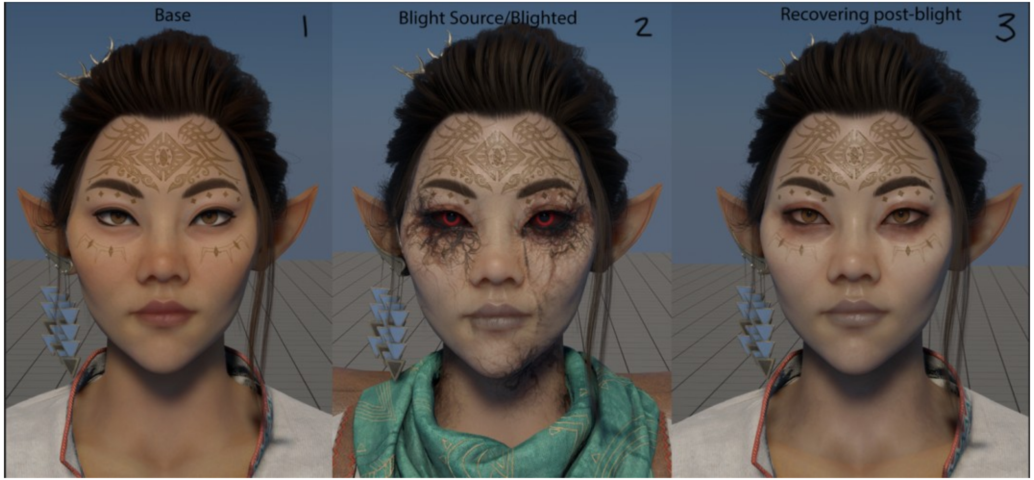
Infill Characters Trade-offs

- Used 2 head blends instead of 3 as in Full NPCs.
- Stop blending computations at long camera distance.
- No feature morphs were used because camera never gets as close.
- Card hair only for infill NPCs.
- Shader params subset was used. For example: no makeup for infill characters.
- Texture sets subset was used for complexion.
- 1-mip offset for all textures on platforms with memory constraints was used.
- Did not make separate low poly models because LODs were usually enough for reducing the geo complexity at that distance from camera.
- Division of data by region allowed to load/unload assets as needed for that region.
    - Also provides better content management for artists

Infill Characters

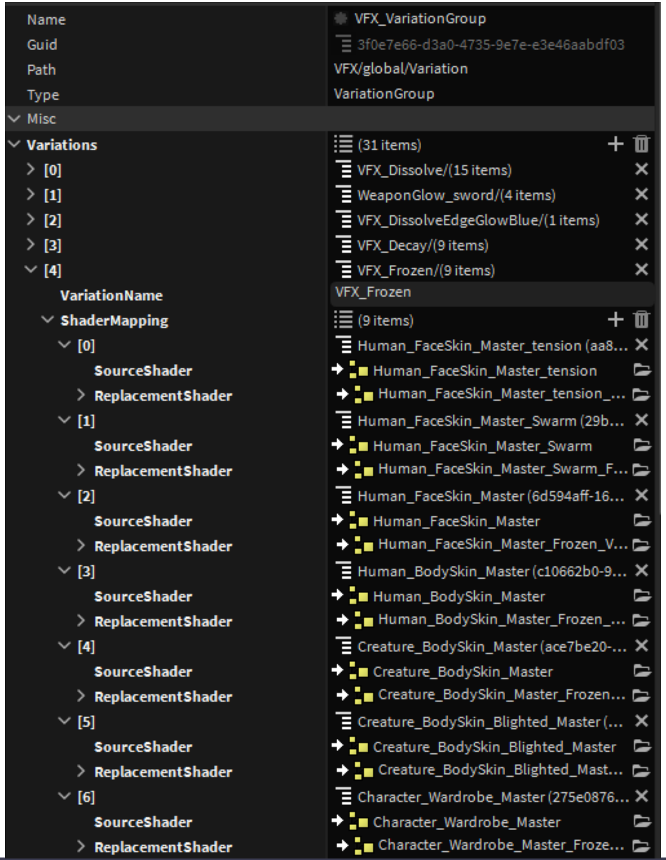
Dynamic Shader Variation System

- Used for Damage Over Time effects like Poison/Burn/Freeze on players and NPCs
- Built to utilize the shader parameter blocks.
- These also can be stacked on top of the other shader customizations.
- Avoids SRV limitations and shader permutations bloat.
- Also useful for showing injuries or blighted states on character.



Dynamic Shader Variation System

- Replacement shader for each shader that needed a variation.
    - E.g.: For Burn effect on body, body skin shader had a body skin burn shader
- All shader for that variation are assigned to a mesh variation with name hash
- Gameplay can then invoke this variation using name hash.



Dynamic Shader Variation System

Pipeline

1. Compare shader values on replacement shader with original shader
2. Store differences in a shader parameter block
3. Create a mesh variation for each shader parameter block
4. Store mesh variation and name hash in database

Runtime

1. Variation is invoked using name hash
2. Look up mesh variations against name hash
3. Apply the shader parameter block from mesh variation

Dynamic Shader Variation System

Gear System

- Used for equipping armor sets, head gear, weapons etc.
- Pre-built armor sets were collection of appearance items.
- Tie-in to inventory system.
- Items have designated slots.

Takeaways

- Start development of character customization early.
- Start with dev tools to see what customizations are required for character design.
- Refine the options based on the impact in game. E.g. nail polish options was abandoned.
- Measure impact vs effort. Efforts for jewelry on head body blend shapes was too high with lesser impact.
- Keep an eye on memory and make adjustments.

Thank You!

Dragon Age: The Veilguard Rendering Team:

James Power
Patrick Chan
Thomas Roy
Wyatt Hammond

Special Thanks To:

Sam Moore(Performance Lead)
Character Art & Pipeline Teams

Frostbite Rendering Team

Q&A