Documentation

Github Wiki Unity Thread UMA Discord Code Documentation

Contents

About UMA

Unity Multipurpose Avatar (UMA) is an open framework for creating avatars, offering base code and example content. You can customize UMA for your projects and share or sell creations via Unity Asset Store. Designed for multiplayer games, UMA includes code to synchronize avatar data between clients and servers. Depending on your needs, custom solutions may be needed to optimize serialized data.

Minimum Requirements

UMA 2.14 requires Unity 2022.3 or above. Alpha or Beta versions of Unity may or may not work, but are not supported until the release version.
UMA 2.14 requires .Net compatibility level in player settings. It will work with both Mono and IL2CPP backends.
If using Blendshapes, GPU skinning will need to be disabled in some versions of Unity.

Performance and Memory Usage

UMA framework provides a set of high-resolution content, flexible enough for generating a crowd with tons of random avatars or high quality customized avatars for cutscenes. Source textures are provided for generating final atlas resolution of up to 8K textures. Depending on the amount of extra content being imported to the project, it might be necessary to handle memory management or reduce texture resolution. You can also operate in various modes that do not generate an Atlas, but do still allow compositing of overlays.

Every UMA avatar created has its own unique mesh and Atlas texture, requiring extra memory. The standard atlas resolution of 4K is recommended for creating a small number of avatars. For games creating on a huge number of avatars, using lower atlas resolutions, or using alternate texture modes (like Use Existing Textures with render-time compositing, or Use Existing Materials) or sharing mesh and atlas data will be necessary.

UMA was initially planned to provide 50 avatars on screen, but the latest version can easily handle more than a hundred unique avatars.

When creating NPC or Crowd characters that do not need to be customizable, it is usually optimal to generate a “Prefab” from them using the Prefab Maker. This will use significantly less memory than duplicating a bunch of customizable UMA’s in the scene.

When using a large number of slots and overlays, with large textures, it is recommended to use Addressables, so UMA can load and unload data from memory.

How Does it Work?

Creating 3d characters is a time-consuming process that requires a number of different knowledge areas. Usually, each character is created based on a unique mesh and rig and is individually skinned and textured.

When developing games that might require a huge number of avatars, it's expected to develop a solution to handle avatar creation in an efficient way. Usually they start from a set of base meshes and follow standards to be able to share body parts and content. Each project ends up with a different solution, and it's hard to share content or code between them.

UMA is meant to provide an open and flexible solution, which makes it possible to share content and code between different projects, resulting in a powerful tool for the entire community.

UMA has two main goals: Sharing content across avatars that uses same base mesh and optimizing created avatars, while providing the ability to change avatar shape in real-time.

To achieve that, a special rig structure was created that handles bone deformation in a strategic way that makes it possible to deform UMA shape based on changes on bones position, scale and rotation.

UMA example avatars are based on two base meshes, a male and a female. Each of them can share clothing and accessories and can be used as a base on the creation of new meshes for different races.

Because clothes and accessories are skinned to UMA base meshes, they receive the same influence of UMA body shape, so any mesh deforms to any UMA body.

This way, if you have an armor or dress, it can be shared between all male or female avatars, even if they have very different body shapes.

An overlay system was implemented that makes it possible to combine many different textures when generating the final atlas. Those extra textures can be used to create even more variation to each avatar, and can be used for clothes, details and many other possibilities.

UMA optimization occurs in many steps: Each UMA avatar can have a unique texture atlas providing all necessary texture data, this makes it possible to have each UMA generating a single draw call, in the case of a single material being used.

All UMA parts are baked together into one final mesh, which reduces the calculations involved in processing. Joen Joensen from Unity team implemented an advanced skinned mesh combiner to accomplish this.

How is Content Created?

See the Content Creation Section

What's new in 2.14

Minimum supported version is now 2022.3

Support for Unity 6 with HDRP.
New welcome window to help with common tasks.
New "Vertex Modifiers" and editor that allow you to create equippable vertex modifications.
Fix for creating multiple slots from single mesh with multiple materials
New bone poses for boy/girl races
Can now assign overlay transforms to shared colors (through the property block).

Async GPU Texture downloader – downloads Rendertextures to Texture2D. The number processed per frame is user definable. Note: like all NativeArrays, this will be slower in the editor than in a build.
UMA Preferences have been moved to project specific settings, and are now available in the Project Settings dialog.
Fixed shared color table editor

Quick start

Example Scenes

View the example scenes. There are quite a few located in UMA\Examples. A good first one to examine is “UMA DCS Demo - Simple Setup”.

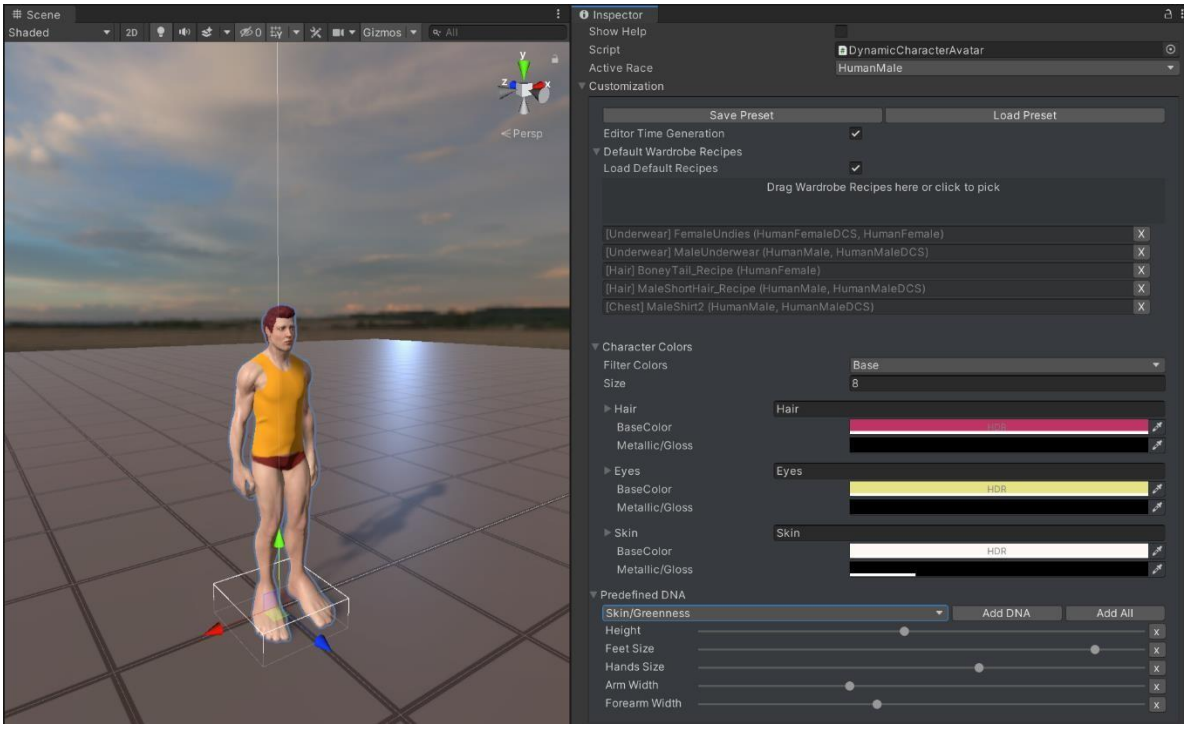
New Scene

1) Open the ***UMA/Getting Started*** folder - this folder contains various prefabs that can be used to get started quickly.

2) Find the prefab "UMA_GLIB" and drag this into the scene. This is the object that contains the UMA context, generator and mesh combiner objects. The context in this prefab goes directly to the UMA Asset Index to access items.

3) Next, find the prefab "UMADynamicCharacterAvatar" and drag this into the scene. This will be an UMA avatar. It contains the script "DynamicCharacterAvatar".

4) Set it's position to wherever you want and make sure there is an object or terrain underneath the avatar object. Also, make sure it is in view of the game Camera. It should generate and display the character, and allow you to make changes to it in the inspector, in the customization section:



5) From here you could change to other races or add new wardrobe items, update colors, add DNA and change it, etc. Note that only wardrobe items that match the current characters race will be loaded, but you can still add other items for other races.

Basic Concepts

UMAGlobalContext and UMAContext

The UMAGlobalContext contains the methods to access the assets needed to generate UMA characters. It accesses these items straight from the Global Library. A prefab that has been setup for most use cases has been provided in the **UMA/Getting Started** folder (UMA_GLIB). There is another context (UMAContext) that accesses assets from a set of libraries. It loads items from various libraries that store references to the items in the scene. This context is kept for compatibility with older code. It is recommended when using UMA 2.10 to use the Global Context.

A Context of some type is required in the scene for the UMA system to work correctly. You can create this one time, and set the flag “Don’t Destroy On Load” to keep it resident – doing this requires you to structure your game in a specific way and is beyond the scope of this document. See https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html for more information.

UMAGenerator

The UMAGenerator component is included on the UMAContext in the prefab. This component is used to tune the generation of UMA Avatars.

FitAtlas	This value should be set true unless you have provided your own Texture Merge prefab. When this is set, textures are reduced as required to make all textures fit into the atlas.
Convert Render Textures	Checking this will convert the generated textures from RenderTextures to Standard Texture2D. This will slow the generation process quite a bit, but will also allow you to programmatically change the texture bits. Setting this is not recommended unless you specifically require this functionality.
Convert Mip Maps	Checking this will generate mipmaps. This is recommended.
Atlas Resolution	This will set the maximum size of any generated texture atlas.
Default Overlay Asset	If an overlay is not found during generation, this will substitute.
Initial Scale Factor	This is the scale factor applied to overlays when they are added to the atlas. Setting a scale factor of 2 will result in all overlays starting with their size divided by 2. This is useful for improving texture RAM usage on lower end systems. This should normally be set to 1, and increased to improve RAM usage.
Fast Generation	If this is checked, UMA will generate the skeleton and mesh for a queued UMA in one frame. If this is not checked, they will be generated in separate frames.
Iteration Count	How many iterations of the generation loop to process in one frame. If Fast Generation is enabled, then this is equivalent to how many UMA's to generate per frame.
Garbage Collection Rate	How many iterations before collecting garbage and freeing memory.
Process All Pending	If you check this, all UMAs will be generated on the next frame.
Texture Merge prefab	If you provide your own texture merge module, it should be included on this prefab. This is an advanced option.
Mesh Combiner	If you provide your own mesh combiner, it should be included on this prefab. This is an advanced option.
Edit Time Atlas Resolution	This is the atlas size that is used during editing. You can make this lower so that generation is faster during editing.
Edit Time Scale Factor	Like the other edit-time option, this is used to tune the edit time generation, and is used in place of Initial Scale Factor during edit time.

Sharper Fit Textures	When textures do not fit into an atlas, they will be resized to fit. Checking this box will tell the generator to use a higher mip map to scale down to the size. If this is not checked, the default is to let the driver decide which mipmap to use, usually resulting in a fit texture that is slightly blurrier.
Atlas Overflow Fit Method	**Best Fit Square** – Using this method, when the atlas is overbooked, the generator will create the largest possible square to contain the the atlas textures, and then reduce the textures to fit in the atlas exactly. This results in the least amount of wasted space and the maximum amount of resolution. This is the default for UMA 2.11. **Decrease Resolution** – Using this method, the system will attempt to reduce the texture sizes on a percentage basis to fit in the atlas. The percentage size to reduce for each fit pass can be specified (previous versions of UMA used this method, with a hardcoded value of 0.5).

Dynamic Character Avatar

The DynamicCharacterAvatar is the highest level component to utilize UMA and the wardrobe system. It allows you to easily select races, set events, character colors, and default wardrobe recipes. A default prefab named ***UMADynamicCharacterAvatar*** is located in the ***UMA/Getting Started*** folder.

Wardrobe System

The Wardrobe system introduces the concept of “Wardrobe Recipes”. These containers allow you to select compatible races that they are to be used on, the wardrobe slot (not to be confused with regular slots) as well as other configuration options. Finally, you can add the various slots and overlays that represent this wardrobe recipe. In essence, this encapsulates the idea of an “item” or “clothing” that can used as one object. Each wardrobe item will contain one or more slots, and one or more overlays for each slot. They can even contain slots that have been used on the base race recipe, or on other wardrobe slots. When this happens, the extra slots are merged out, and the overlays are added to the existing slot. For example, if you want to have a tattoo on the face, you would add a face slot, and a tattoo overlay – even though there is already a face slot (with the face overlay) on the base race. When the character is built, the extra face slot will be removed, and the tattoo overlay will be added after the face overlay.

Changing the avatar in code

To set a wardrobe recipe in script, all you need is the function:

DynamicCharacterAvatar.SetSlot(UMATextRecipe "wardrobe recipe")

This will attempt to apply the named recipe (looks for compatible race and appropriate slot). After setting the wardrobe recipe, the changes will not take affect until you call:

DynamicCharacterAvatar.BuildCharacter();

This is so the user can make several changes before attempting to rebuild the UMA.

Finally, to clear a wardrobe recipe, simply call;

DynamicCharacterAvatar.ClearSlot(string "wardrobe slot name")

The "wardrobe slot name" is the location that the recipe is set to, for example "chest", "head", or "hands".

To set a character color on your avatar use;

DynamicCharacterAvatar.SetColor(string ColorName, Color colorValue)

This version of SetColor will only set the color that is multipled by the albedo. To have more control over how the color is set, you can call SetRawColor:

DynamicCharacterAvatar.SetRawColor(string ColorName, OverlayColorData colors);

When using this, you must construct an OverlayColorData that contains a channel (Both multiplied and additive) for each texture in the overlay.

As with wardrobe recipes, colors are cached, so for changes to take affect you will need to call;

DynamicCharacterAvatar.UpdateColors(bool triggerDirty)

*note-set triggerDirty to true for immediate results, otherwise the colors will be set on the next "buildCharacter" call

Saving and loading from a string

You can save the DynamicCharacterAvatar to a string using an AvatarDefinition.

```
compressedString = Avatar.GetAvatarDefinition(true).ToCompressedString("|");
```

Then load him from that string:

```
AvatarDefinition adf = AvatarDefinition.FromCompressedString(compressedString, '|');
Avatar.LoadAvatarDefinition(adf);
```

```
Avatar.BuildCharacter(false); // don't restore old DNA...
```

Preloading DNA

You can preload DNA on a DynamicCharacterAvatar (DCA) by creating a UMAPredefinedDNA object, and adding it to your DCA after you instantiate it, but before you build it. This is not a Unity Monobehavior or ScriptableObject – it's a straight C# class that holds string/float pairs for the DNA. This DNA is loaded into the character during BuildCharacter().

To use this class, simply create a new instance, fill it with DNA and Values (using UMAPrefinedDNA.AddDNA(string,value), and then assign it to the DCA.predefinedDNA field after instantiation.

Random Generation

UMARandomAvatar

The UMARandomAvatar is a component to generate a Random DynamicCharacterAvatar. This component can also be used to test the generation of Avatars by generating an array of Avatars centered around the game object.

To create a Random Avatar, create a new Game Object, and place the UMARandomAvatar component on it.

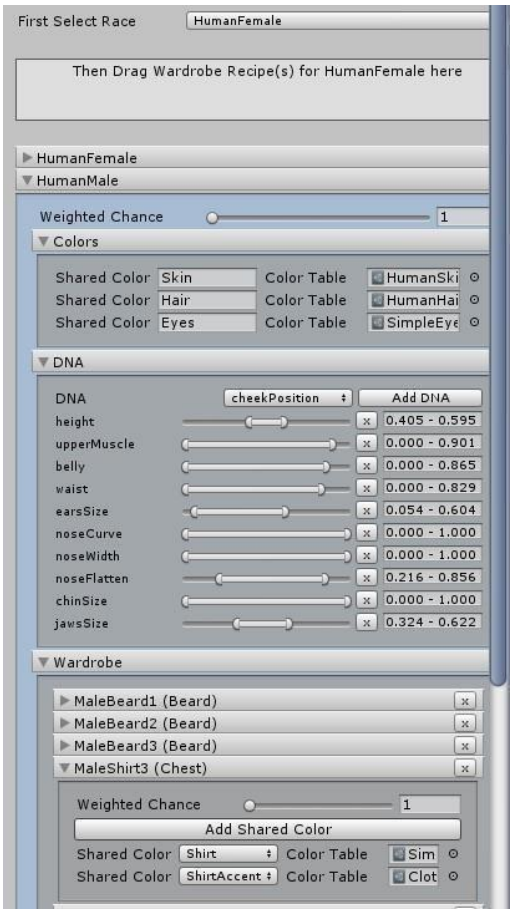
The UMARandomAvatar uses an object called a UMARandomizer to choose how to generate the object. See the documentation for this object below.

You can have as many UMARandomizers on a Random Avatar as you like – one will be randomly chosen. This allows you to specify different sets of random data to fine tune the generation – for example, make sure darker skin colors are matched with darker hair colors.

UMARandomizer

This is a scriptable object that defines the random properties for a character. A RandomAvatar can contain many UMARandomizers, but only one is chosen and used to generate a character. The UMARandomizer allows you to setup random data for specific races (for example, human male, human female). To use the randomizer, select the race you want to work on at the top in the inspector, and then drop all of the wardrobe assets for that race on the drop area. This will create the random race instance, and allow you to set the colors, chances, DNA, etc for that race.

Chances on each item default to 1. (there is a 1 in N chance of selecting that item – with N being the total number of “chances” for each similar item). For example, if you had 12 items, and each item had 1 chance, then the chance to select an item is 1 out of 12. If you set an item to 5 chances, then every item except that item would now have a 1 in 17 chance, while it would have a 5 in 17 chance of being selected.



Libraries and Global Library

The different UMA components are assembled at runtime. To find these, UMA can use several different methods.

UMAGlobalContext and Global Library

The new method in 2.10 is to use the UMAGlobalContext, and access the Global Library directly. It is recommended to upgrade to this method in 2.10 and above. However, the older versions are still in UMA for backwards compatibility.

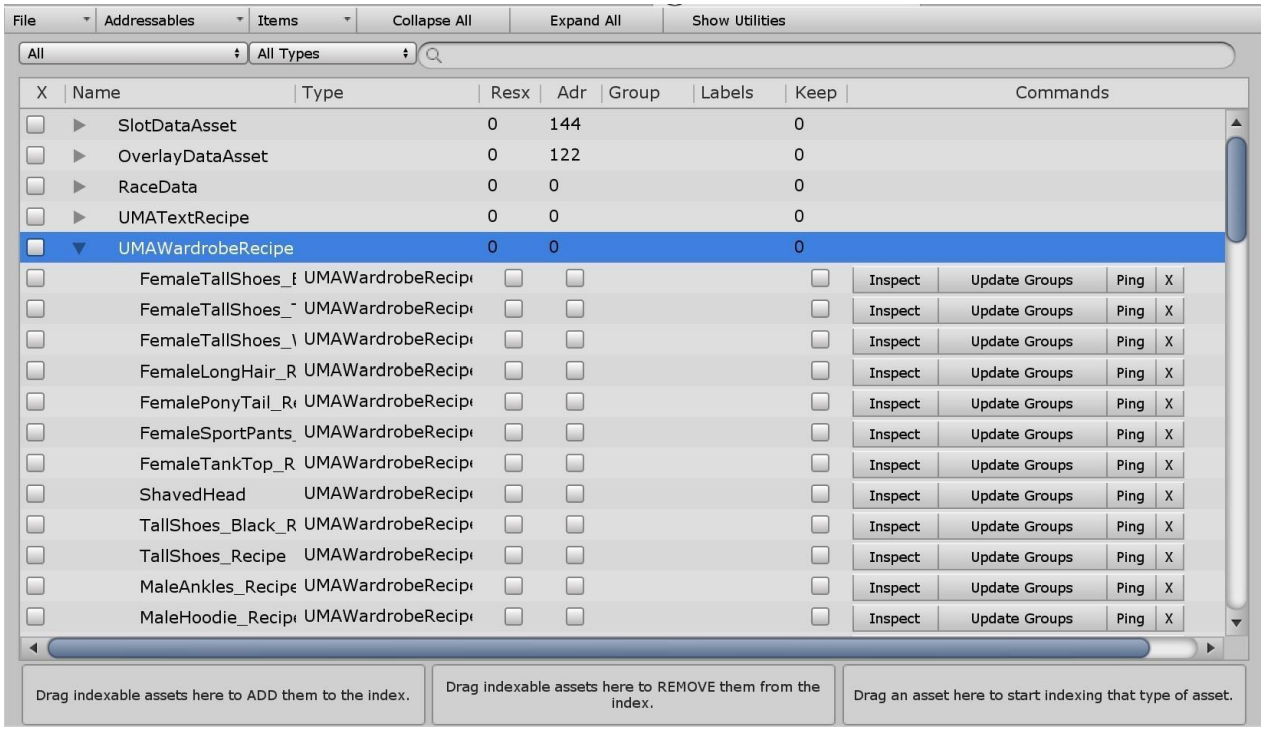
UMAContext and Scene based Libraries

The original methods from 1.0 and 2.5 are using the scene-based libraries - OverlayLibrary, SlotLibrary, DynamicOverlayLibrary, DynamicSlotLibrary, etc. These libraries are only valid for the scene they are in. To add items to these libraries, select them in the scene hierarchy, and drop items onto drop pad for the specific library components in the inspector. These libraries will not be supported in future versions. The Dynamic versions will attempt to build the libraries on the fly using the global library (asset index). These will be removed in future versions of UMA.

Global LIbrary

To view the global library, use the menu item UMA/Global Library Window to open the Library window, and drag/drop your folders containing the assets onto the drop pad. The library window shows all the indexed items by type (you can expand a type by clicking on it). You can inspect an item by clicking the ***Inspect*** button. You can select the item in the project window by clicking on the ***Ping*** button. You can filter the window by
If you need to access the asset, you can click the name of the asset, and it will be selected in the project. In addition, you can filter the global library by using the filter above the indexed types.

When using Addressables, the ***Adr*** column shows the if the item is addressable. The ***Group*** item shows the group it was added to, and the ***Labels*** column shows the labels that were added to the item. Items are labelled by the recipes they are in, so they can loaded by label in a logical manner. The ***Keep*** flag can be toggled on to tell the system not to delete this item when clearing orphaned items. The ***Update Groups*** button is used to update the addressable groups with the recipes contents. You should use this if you have changed or added items in the recipe and do not want to regenerate.



The three drop pads on the bottom are used to add and remove items from the menu, and to add types to the index. To add or remove items, drop the items (or folders) onto the pad, and they will be scanned for indexable items and added. If you want to include your own items in the index, drop any item on the last pad, and a new type will be indexed.

Menu Items

File/Rebuild From Project: The project is scanned, and every indexable item is added to the index.

File/Repair and Remove Invalid Items: Rebuilds the index using the GUID and/or location of the asset, updates the items with the correct flags, etc. This is helpful if you have moved assets around and/or updated them outside of Unity. If an item cannot be found, it is removed from the index.

File/Toggle Utility Pannel: This enables or disables the utility panel, which has some utilities for updating the items en-masse.

File/Empty Index: Removes everything from the index. This should be used only if you plan to recreate the library from scratch.

Addressables/Generate/Single Group (fast): This adds everything in the library into a single addressables group that is setup to “pack separately”. This option will produce an asset bundle for each item. Use the options in the UMA preferences to change how this group packs and labels items.

Addressables/Generate/Generate Groups (Optimized): This will generate a group for each recipe. Any item that is in one or more recipe will move into the shared group, which is packed separately. Using this option will produce the minimum number of Asset Bundles and IO, but takes longer to calculate.

Addressables/Delete Empty Groups: This will remove any empty groups from the addressables system.

Addressables/Remove orphaned slots: Use this only after building groups. This will remove any slots that were not added to addressables, were not marked “keep”, or were not forced to be included in resources (in the recipe).

Addressables/Remove orphaned overlays: Use this only after building groups. This will remove any overlays that were not added to addressables, were not marked “keep”, or were not forced to be included in resources (in the recipe).

The ***items*** menu contains various functions for selecting, deselecting and removing items.

Race

A "Race" in UMA is a way to specify the information that is specific to a single archetype. This includes TPoses, DNA Converters, Slots, Expressions, and wardrobe slots. For example there are RaceData entries for Male Humans and Female Humans, because they have slightly different ***TPoses*** and gender specific DNA converters, despite sharing the same DNA types.

The term "race" can be confusing in UMA. It can be used for avatars that share the same base mesh and DNA. For example, if you have a dwarf that is created from the human base mesh simply by changing the dna to be shorter and stockier, then it will still be the same "race" as the human, in UMA terms. This is completely abstract from any game level concept of character races. This is the reason why a human male and a human female are different "races" in UMA, because they have different base meshes. A Race can also define a “Base Recipe” and a set of Wardrobe Slots. The Base Recipe defines how your avatar looks without any wardrobe items applied. Wardrobe slots are used on the avatar to hold a single specific wardrobe recipe. Click here for more information on Races

DNA

The Dna of a UMA avatar are the possible changes or deformations that can be made to customize an avatar. While these are meant to be accessed programmatically, These individual items can also be accessed and changed at runtime to see immediate results on the "UMAData" component.

Click here for more information on DNA

Slots

A “Slot” is a representation of a mesh. It contains bones, vertexes, weights, etc. Slots are basically containers holding all necessary mesh data to be combined into an UMA avatar, pre-extracted from a skinned mesh.

For example the base meshes provided are normally split into several pieces, such as head, torso and legs... and then implemented as slots which can be combined in many different ways. An UMA avatar is in fact, the combination of many different slots, some of them carrying body parts, others providing clothing or accessories. Lots of UMA variation can be created simply by combining different slots for each avatar.

Slots also have a material sample, which is usually then combined with all other slots that share same material, if you are using atlassing. Female eyelashes for example, have a unique transparent material that can be shared with transparent hair. It's necessary to set an UMAMaterial on all slots, as those are used to consider how meshes will be combined. In many cases, the same UMAMaterial can be used for all slots.

The big difference between body parts and other content is that body parts need to be combined in a way that the seams won't be visible. To handle this, it's important that the vertices along mesh seams share the same position and normal values to avoid lighting artifacts.

To handle that, we provide a tool for importing meshes that recalculate the normal and tangent data based on a reference mesh.

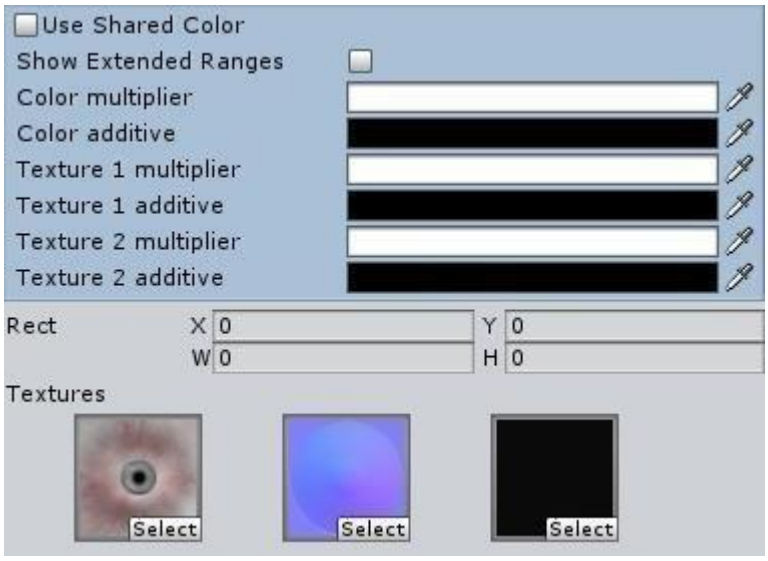
Overlays

Each slot that contains a mesh requires at least one overlay set but can contain more than one. Overlays carry all the necessary textures to generate the final material and might have extra information on how they are combined. The first overlay in the list provides the base textures, and all other overlays included on a slot are combined with the first one, in sequence, generating the final texture or texture atlas. Overlays are combined using the alpha channel of the first texture in the overlay, or you can supply an alpha mask texture to override the alpha channel.

Colors

Each overlay can modify its color multiplicatively or additively. This can be used to tint overlays, such as having a skin texture that can be tinted to different skin tones.

Overlays can be layered on top of each other to selectively color parts of the final output.



When using Atlas/No Atlas type materials, wach texture specifies what blending mode is used to combine the textures.These blendmodes are Normal, Multiply, Overlay, Screen, Darken, Lighten, ColorDodge, ColorBurn, SoftLight, HardLight, and Subtract. These should produce the same results as photoshop blend modes.

Shared Colors

Shared Colors can be set as a common color to be used in an overlay set to use it. This way the same color can be used across overlays. For example, being able to set the same color on all skin textures (eg, body and face, etc…). They are set by creating a Shared Color at the top of your recipe. Then in the overlay, toggle “Use Shared Color” and select the shared color channel you set at the top of your recipe. Important: The DynamicCharacterAvatar is in charge of the shared colors – the color on the avatar controls what is plugged into those colors at build time. This ensures that all instances of those shared colors are the same.

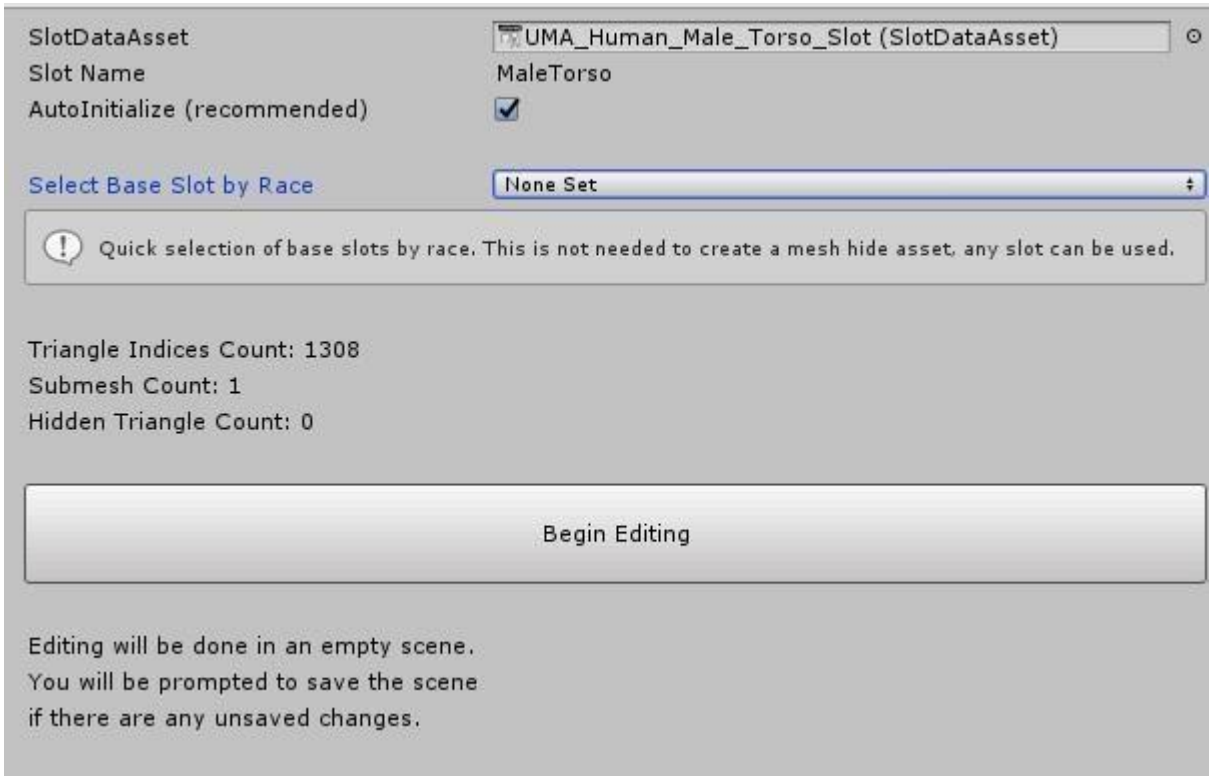
Mesh Hide Asset

The Mesh Hide Asset object is assigned to a Wardrobe recipe to hide portions of the geometry of **other** objects (either parts of the base race slots, or parts of other wardrobe items). This object is used to fix “poke through” when the item is equipped.

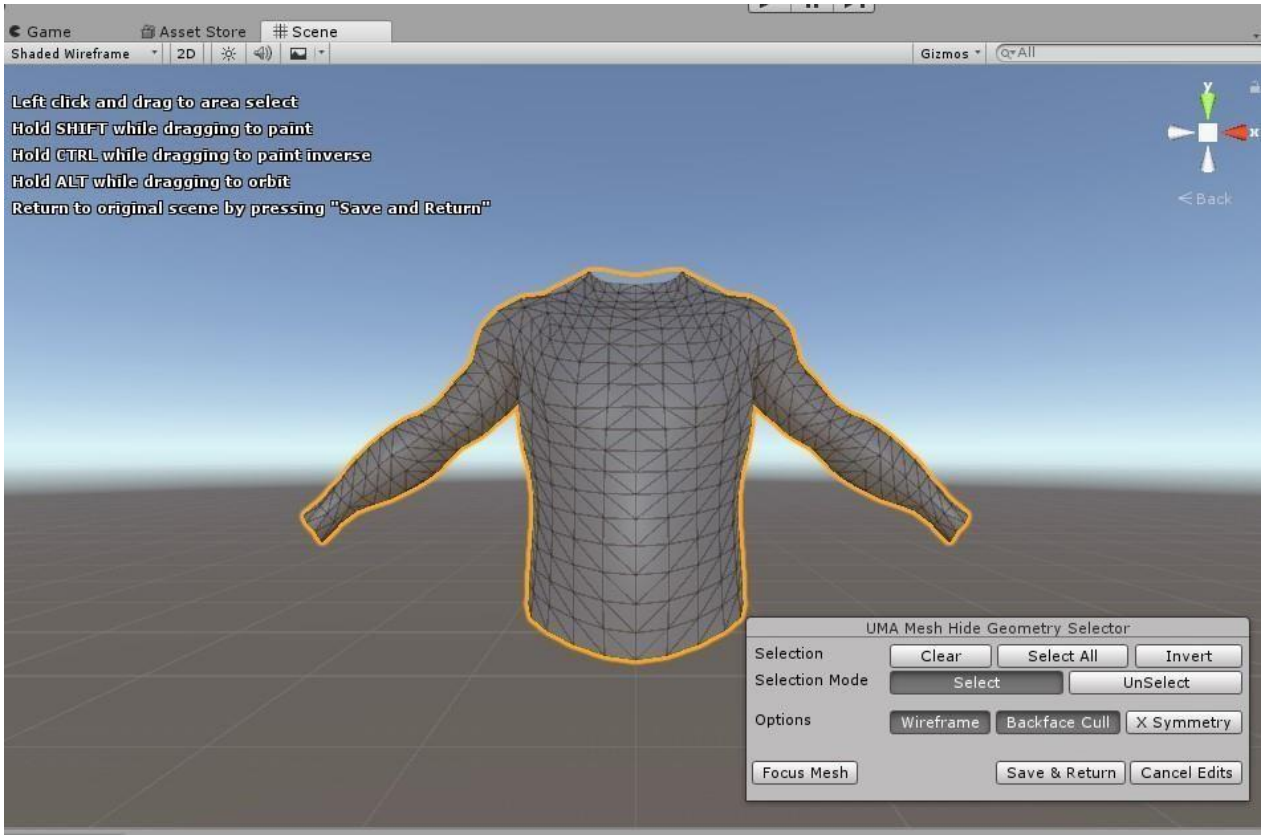
Note: Wardrobe items that completely cover base slots or other wardrobe items should use the functionality in the recipe to hide or suppress the hidden slots instead, as that is more efficient. Mesh Hide Assets are used when the slots are not completely obscured.

To create a Mesh Hide Asset, right click in the project, and select “Create ./ UMA / Misc / Mesh Hide Asset” from the popup menu.
In the inspector, select the SlotdataAsset that you want to be partially obscured. You can either use the select from the field menu, drop a SlotDataAsset on the field, or you can select the race and base race slot below the field.

After selecting the slot, the “Begin Editing” button will be enabled. Press this to load the slot into an empty scene for editing:

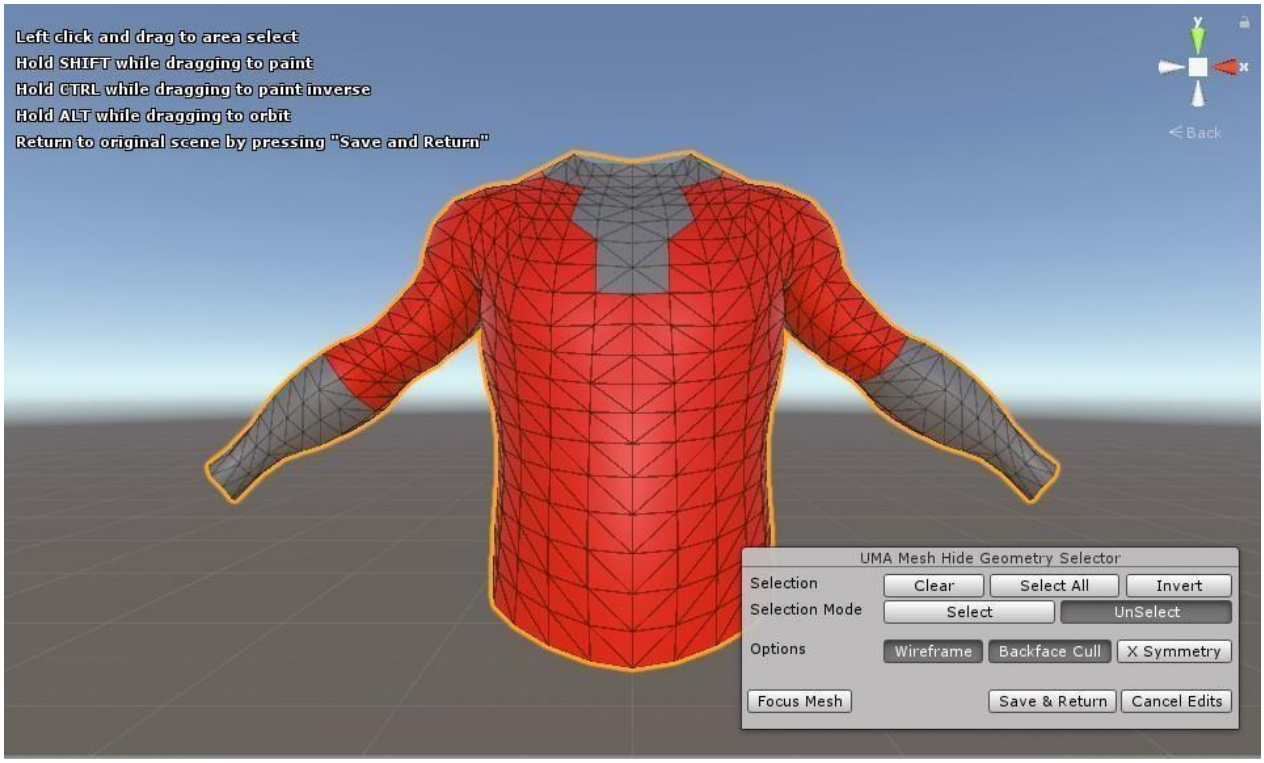


A new scene will open, and will look similar to the following. This scene allows you to select or unselect specific polygons:



Selected polygons are red. These are the polygons that will be hidden. Unselected polygons are gray. These polygons will remain and be visible.

For example, look at the following picture. The forearms and neck will be visible, and all other polygons will be hidden:



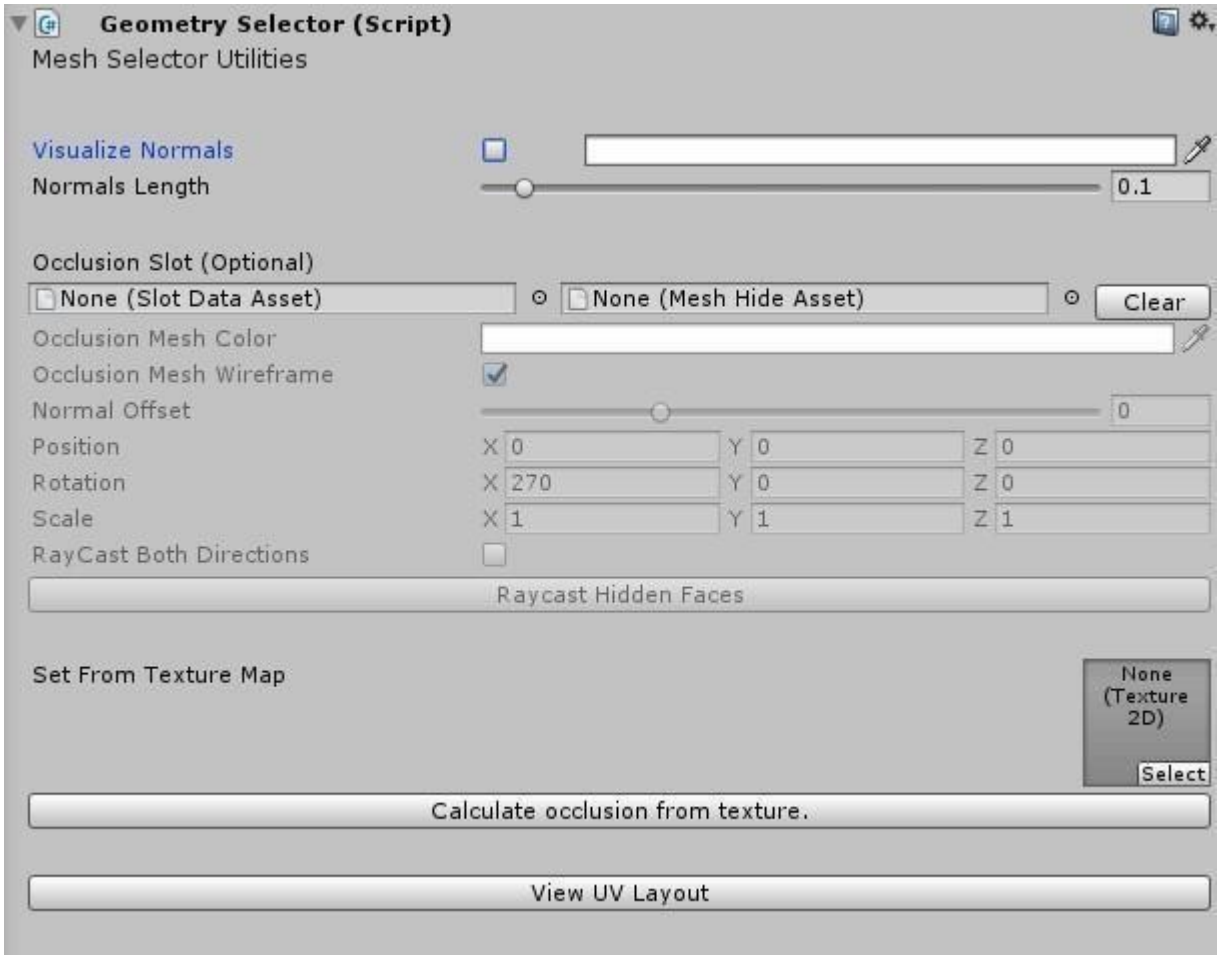
Pressing “Clear” will deselect all of the polygons. Pressing “Select All” will select every polygon. Invert will invert the selection status of all polygons (selected polygons will be unselected, etc.)

You can click on the mesh to select or deselect a polygon. You can drag a bounding box around polygons, and it will select or unselect based on the Selection Mode. You can hold down the SHIFT key to “paint polygons freehand” using the current Selection Mode.

X Symmetry is an experimental drawing mode that works well when you have X mirrored meshes. It attempt to automatically select polygons on the opposite side left/right when selecting or painting polygons.

Press “Save & Return” to finish editing the Mesh Hide Asset and return to the previous scene.

You can choose to automatically select occluded polygons by selecting a an occlusion slot in the inspector. Press “Raycast hidden faces” to detect and select occluded polygons. Sometimes the occlusion slot can be rotated or offset incorrectly. You can adjust the transform of the occlusion slot in the fields provided:



Bone Poses and Physiques

The new Physique slot is intended to allow you to generate and apply your own physiques. To create a Bone Pose Set, right click in your project, and select ***UMA/DNA/Physique Bone Pose Set***. Be sure to name your new set something unique (for example: “Trollface”). This gives you the option to create a wardrobe recipe as well as add the generated assets to the global library. After generating, you will need to define the Bone Pose by selecting it in the project, and editing it in the inspector. To edit the pose, you must be running a scene. Select anything with a UMAData (for example, a DynamicCharacterAvatar), and drop it into the ***Source UMA*** field.

UMA Material

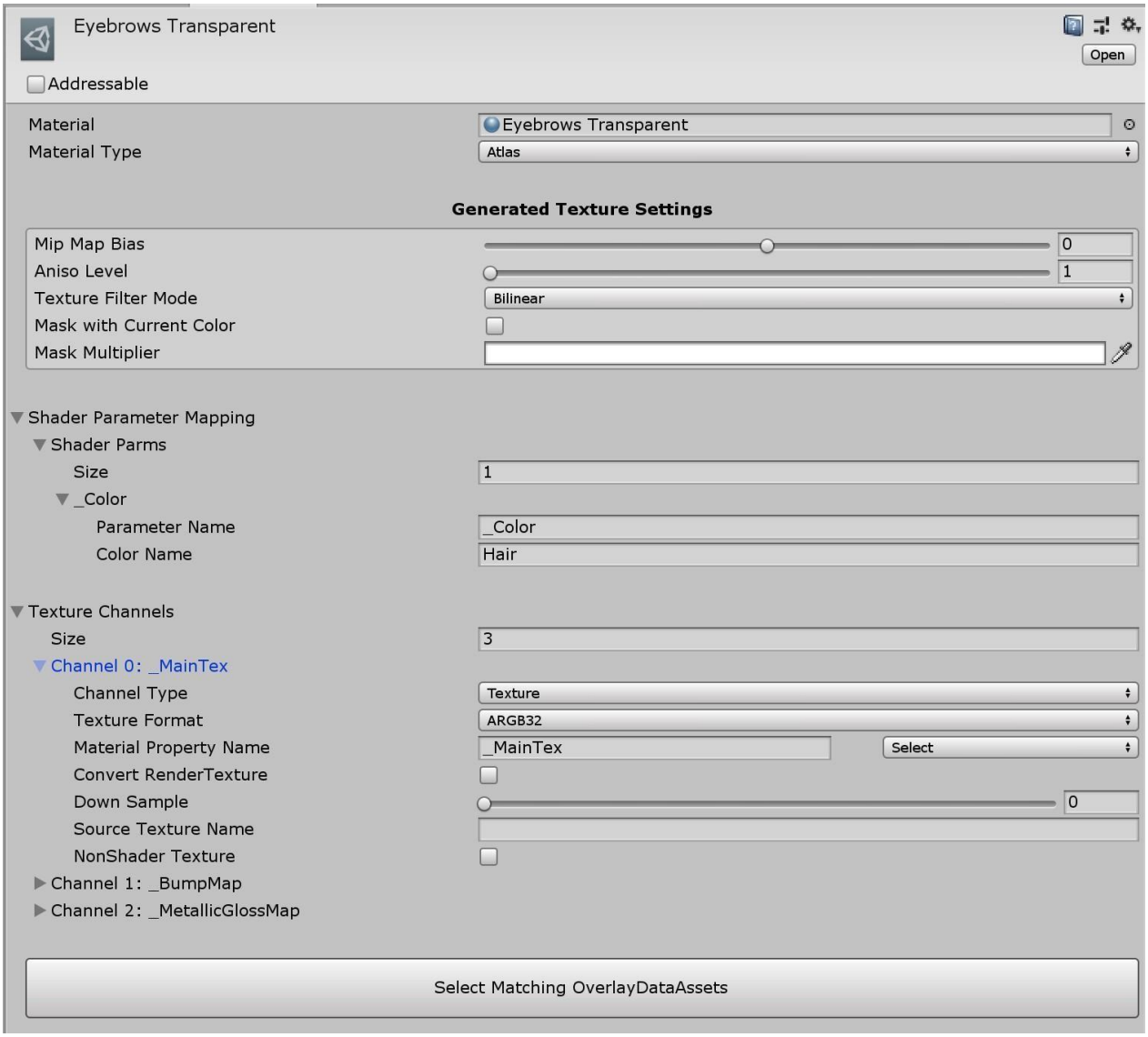
The UMA Material asset object is used by Slots and Overlays. This asset is a wrapper for Unity Materials so that the UMA system can properly group identical materials to be atlassed and merged. An UMAMaterial can be created for any type of material.

There are several prebuilt UMAMaterials for most of the commonly used material and shaders, located in UMA/Content/UMA_Core/HumanShared/Materials.

For example, in the "standard" folder;

"UMA_Diffuse_Normal_Metallic" - Unity Standard Shader that accepts albedo, normal, and metallic/roughness texture.

"UMA_Diffuse_Normal_Metallic_Occlusion" - Unity Standard Shader that accepts albedo, normal, metallic/roughness, and ambient occlusion texture.



Material: The template material used to generate the end material.

Material Type: When the Material type “Atlas” is selected, The UMAMaterial is used to determine what items go together on an atlas, and to specify the unity material that is used to generate the material for the atlas.

Generated Texture Settings: You can adjust the various setting for the generated textures. Mask with current color specifies what color is used as the base color for alpha and cutout

albedo textures. This can give a better result when generating hair textures, etc. Mask Multiplier is used to darken the masked texture to better match the source textures.

Shader Parameter Mapping:

This is used to map shared colors to shader parameters in the generated materials. The shared color must be present in the recipe. The current value of the shared color is used during the build process to update the material. This can be used, for example, to set color parameters for skin and hair shaders.

Click here for more information on UMA Materials

UMA Simple LOD

UMA comes with a Simple LOD solution. This solution can do both texture and slot level LOD. Parameters for the UMASimpleLOD script are:

LOD Distance	The distance to step to another LOD. The default is 5.
Swap Slots	Check this to use Slot level LOD also.
LOD Offset	Used to offset the LOD chosen by distance for slots. Subtracted from the calculated value. For example, if this is set to 1, then LOD level 1 for slots will not be chosen until actual distance is 2*LOD Distance.
MAX LOD	The lowest available LOD to look for if an LOD slot is not found.
Max Reduction	The maximum scale reduction for the Atlas textures. (The default, 8, means the texture can be reduced in half 8 times)

Texture LOD is automatic. Each time the avatar is a multiple of the LOD distance, the atlas texture is reduced in half. For example, at LOD level 3, The texture is reduce to 1/8th the maximum size.

Slot LOD happens only if the “Swap Slots” option is checked. This requires that you pregenerate slots for the various LOD levels, reducing polygons for each level. The number of LOD levels are variable. If an LOD level is not found, it will use the previous level.
The slots are found in the library by name, identified by the LOD level assigned. For example, if you had a slot named “vest” and you wanted two LOD levels for it, you would create two additional slots, named “vest_LOD1” and “vest_LOD2”. These should be in the global library, or an asset bundle.

Texture LOD and Slot LOD happen at mid-level. The entire character is not regenerated from base race and wardrobe slots. Instead, the existing recipe is updated, and the character is reset to generate. This is faster than generating the character from scratch.

Low Level access

DynamicAvatar

It is possible to use UMA at a lower level, using a DynamicAvatar. A DynamicAvatar does not have a base race recipe, wardrobe, or shared colors. This avatar is used to construct a character straight from a UMATextRecipe. To change the Avatar, you can access the low level slots and overlays on the UMAData, or you can simply create a new avatar and provide a new recipe.

Text Recipes

Text Recipes for the DynamicAvatar are created by right clicking in the project, and selecting Create/UMA/Core/Text Recipe. This will create a UMATextRecipe. You set the “race” on the slots tab, after which you can press the “Add DNA” button to copy the DNA to the avatar to adjust. Since this is for the DynamicAvatar, you will have to add all the parts of the race yourself – they are not copied from the base race. Add the slots and overlays, and set the colors. Once you have added DNA, you can switch to the DNA tab, and modify the DNA for the character also.

Place the recipe in the DynamicAvatars “UMA Recipe” field. You can also add additional utility recipes – such as a capsule collider recipe, and attach a runtime animator controller.

Content Creation

Animations

UMA does nothing special with animations so any generic or humanoid animations can be used with the appropriate rig.

Clothes

Making new clothes for a race involves taking a mesh, skinning it to the target character’s skeleton, then importing it in to unity and creating slots, overlays, and a wardrobe recipe for it. The first step will be very dependent on the modeling software you use.

-Common across any modelers though, you’ll want to import your target race or character mesh into a scene.
-Deform your cloth to fit how you want it to fit to your race in its neutral position.

-Add a skin modifier to your cloth and skin it to your race’s skeleton. A skin wrap is a good tool here.
-Finally, export the whole skeleton and the cloth mesh as an FBX to import in to unity.

Races (Base Mesh)

A whole new race can range from simple to complex depending the features to support.

Without supporting runtime bone deformation for a race, then a skinned mesh with a valid humanoid skeleton is almost all that is needed.

Both a “unified” version and a “Separated” version of the race are recommended (though not needed). The unified version is a single connected mesh. The separated version is with cut, separate mesh for individual body parts that can be set later in uma to be individually hidden or not. For example, separate mesh for head, torso, hands, legs, feet, etc…

The unified version is then used when building slots as a “seam” mesh, which means the normals from it will be used instead of from the individual cut up meshes. The cut up meshes tend to distort the edges of the mesh and then will not look correct when lined up with their neighbors.

To create a valid new race, you will need to create a RaceData asset, a base recipe for that race, and a T Pose asset.
The base recipe is a standard TextRecipe of all the race’s default slots and overlays. This will be added to the corresponding field on the RaceData.
The T Pose asset is extracted from an FBX of this race with it’s full skeleton. After configuring the Mecanim Avatar that is standard in Unity, then you can use the UMA dropdown function “Extract T Pose”. This will create a new T Pose asset for your race that you can add to the corresponding field.

Click here for more information on RaceData
Click here for more information on TextRecipes
Click here for more information on TPose assets

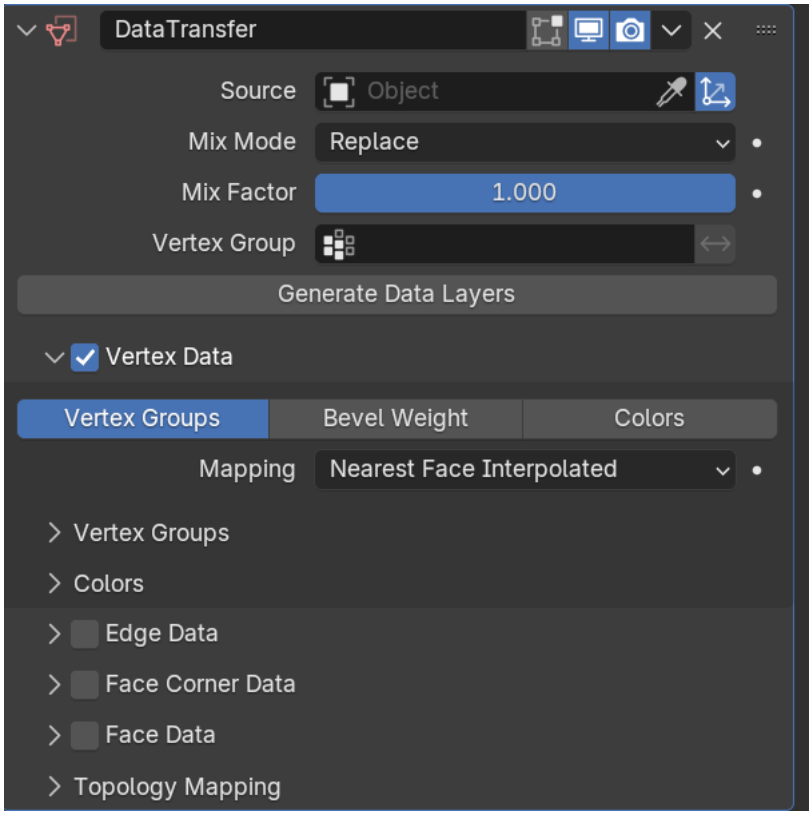
Using Blender to create content

First, download the blends from the content pack repo here:

https://github.com/umasteeringgroup/content-pack/tree/master/ContentPack_2.7.0.0/Blender_28

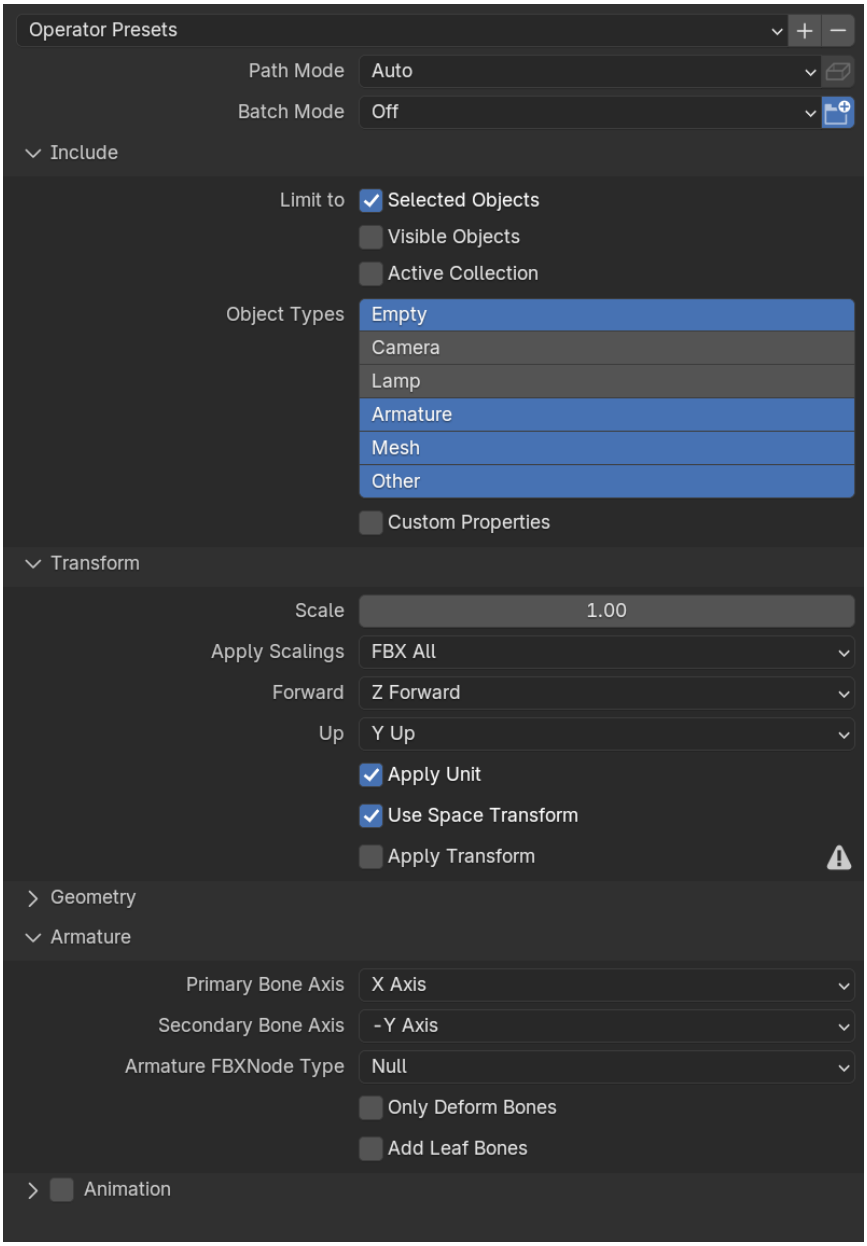
When rigging your clothing, it’s simplest to fit it to the specific model, then copy the rigging information using a “Data Transfer” modifier. After fitting the model, add the modifier, select the source model (the original UMA mesh), select Vertex (you want to copy per vertex data), and “Nearest Face Interpolated”. The select “Vertex Groups”. Make sure Mix Mode is set to “Replace” and press the “Generate Data Layers” button. Then press “**Apply**”. (if you don’t press

Apply - your weighting will be lost). Your clothing should now be rigged. Of course you may need to tune the rigging using Weight Painting.



Export Settings for Blender 4.2+

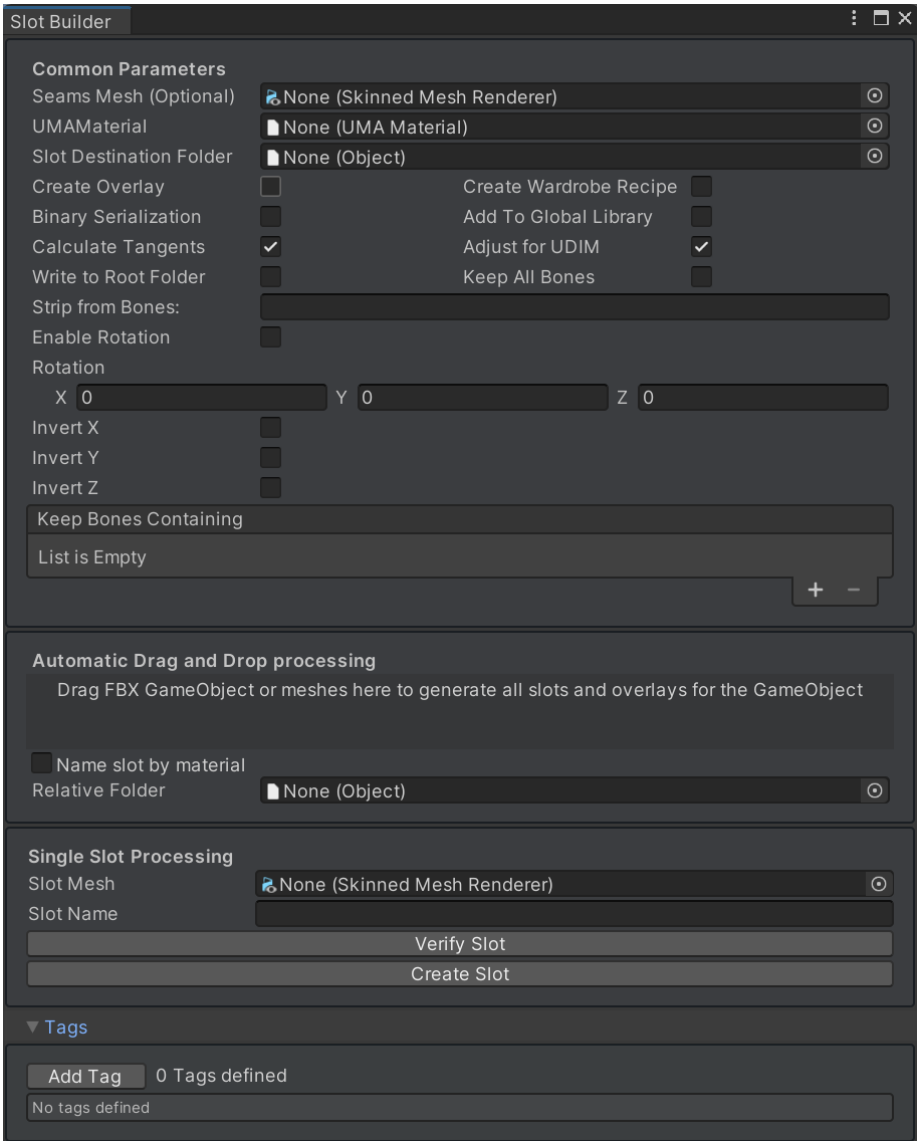
The following settings are used for Blender 4.2+. Be sure you have selected both the armature and the mesh before selecting to export the FBX.



When using these export settings, you should not have to update the scale parameter in Unity.

Using the Slot Builder

Open the slot builder using the UMA/Slot Builder menu item and dock it somewhere. The Slot Builder is used to create the “Slot” from the mesh. To use the slot builder, you’ll fill out the fields and press the “Create slot button.



Seams Mesh: The “seams mesh” is used when you want to fix the normals when you’ve split your mesh into multiple pieces. This is only needed if you have split your mesh up in the modeler, and the modeler has modified the edge normal. If you have a single mesh, it is not needed, nor is it needed if your modelling tool does not modify the normals when you split a mesh.

UMAMaterial: This is the material you want your slot to use. Press the selector button (the small circle to the right) and select the material. Most items use the "UMA_Diffuse_Normal_Metallic" material, But you can select any one you want that you need. When using the "Atlas" mode of the UMAMaterial, Slots that share a material have their textures built into the same texture atlas.

Slot Destination Folder: This is the location where UMA will generate a folder to contain your new slot (and overlay/recipe, if selected). The new folder for your slot will be named the same as the element name (see below).

Create Overlay: Check this if you want an empty overlay created for your slot. Once created, you would need to select it in the project view and add your textures and material to it. The overlay will be named ***{slot name}_Overlay***.

Binary Serialization: When this is selected, the slot will be serialized in binary format. This is faster to load. Uncheck this if you need the asset serialized to text.

Calculate Tangents: This will calculate the tangents for the mesh on load if needed.

Write To Root Folder: Normally, slots are written to an individual subfolder named after the slot. Checking this will instead write the new assets directly to the destination folder.

Create Wardrobe Recipe: Check this to create an empty wardrobe recipe for your slot. Once created, you will need to add your slot(s) and overlay(s) and do the normal setup.

Add to Global Library: Check this if you want all your new items to be added to the global library. If you forget this or something goes wrong, you can just drop the entire folder onto the global library to add them.

Adjust for UDIM: If the UV Coordinates are outside of 0..1 range, this option will move them to the 0..1 tile space. This is for when you have used UDIM textures, and the texture tile is not the root tile.

Keep All Bones: Normally, UMA will discard any bone that is not referenced by the slot, or is a parent of a bone that is referenced by the slot. This will tell UMA not to discard ANY bones. Usually this is overkill, and you should use the "Keep bones containing" list.

Enable Rotation/X/Y/Z/Invert X/Invert Y/Invert Z: These are experimental options that will attempt to rotate the mesh into the correct coordinate space. Do not use these.

Keep bones containing: You can use this list to force UMA to keep any specific bones it might otherwise want to discard. Any bone with the name containing any of the strings will be kept. For example, if you have helper bones named "LHAND_HELPER" and "RHAND_HELPER", you can force UMA to keep both of these by adding "HELPER" to the list.

Automatic Drag and Drop processing:

Drop an FBX here to have UMA create slots for every mesh in the object.

Name slot by material: When this is checked, the new slot will use the Material name for the mesh, instead of the mesh name.

Single Slot processing

Slot Mesh: This is the actual mesh data. Expand your fbx in the project view, and you'll see multiple components below it. Select your clothing mesh, and put it in the Slot Mesh field. You may see a warning that it's too small - if so, go to the import options for you fbx, and set the scale to 100 and apply it, and then drop it on there again.

Create Slot: Press this button to create the slot and any optional items. The folder will be created, the items generated and optionally added to the library. Depending on what you've selected and the size, it could take several seconds to complete.

Verify Slot: This is used to verify that all UV coordinates are in the correct range (0-1). Textures that are atlassed cannot use tiled textures.

UMA Addressables

Overview

As of UMA 2.10, raw asset bundles are no longer supported automatically. If you want to use raw asset bundles, you should download the bundle, read all the items from the bundle, and add them to the Asset Index using ***UMAAssetIndexer.Instance.ProcessNewItem(theItem);*** Items have to be processed before they can be used in any character.

Instead of raw asset bundles, UMA 2.10 uses the Addressables system to manage asset bundles. Using this system, it can manage the load and unloading of assets as needed.

What Happens When An Avatar is built.

A DynamicCharacterAvatar requests assets when it is built using BuildCharacter(). If the character has "BundleCheck" enabled (default is on if you are using Addressables), then after the recipe is generated and the list of slots and overlays are known, it gathers the labels for the character, and passes them to the addressables system for loading asynchronously, and then returns. Once the assets are loaded, the build process schedules the UMA to be generated, and the previously used items are unloaded, freeing up memory (assuming no other UMA has a reference to them). This saves an enormous amount of memory.

Setting up the system to use addressables

If you are not already setup to use addressables for your application, you will need to do so now. Open the package manager, and import the addressables package. You must import the latest version of the addressables package for your unity version. Once the package is imported, open the “Addressables groups window”, and it should have a button on it to generate the addressables data. Push the button to generate the data.

Once that is done, you should make sure you have added all of the recipes you want to use to the global library and then generate the Addressables Groups from the Addressables menu in the global library window. Use the generate single group option during development.

Once the groups are generated, you should switch back to the Addressables Groups window, and build your bundles, and select the play mode. For now, just build your bundles with Build/New Build/Default Build Script. Set the play mode to “use asset database”.
As you get farther in development, you will probably want to change that to “Use Existing Build” so you can test that your groups and bundles work correctly.

Optimizing the usage of addressables

Caching remote items: You can download items from your server using the Addressables.DownloadDependenciesAsync function. See the “Preloader.cs” script for usage. This will cached the items locally, so they are faster to load.

Preloading items for faster access. You can preload items into memory using UMAAssetIndexer.LoadLabelList({labels}); This will load the items into memory, and add a reference to them in the Global Library. This function returns an AsyncOperationHandle<Object> that can be used to determine when the load is complete, and can also be used to unload the items if needed. If you have default recipes that are always in use, you might want to preload them by passing them in the list of labels.

Disabling BundleCheck. If you have preloaded everything you need, you can skip the bundle check on the character by unchecking “Bundle Check”. This will immediately schedule the character to build instead of delaying while the items are loading. You must ensure that ALL items used by the character have been loaded.

Mesh Modifiers

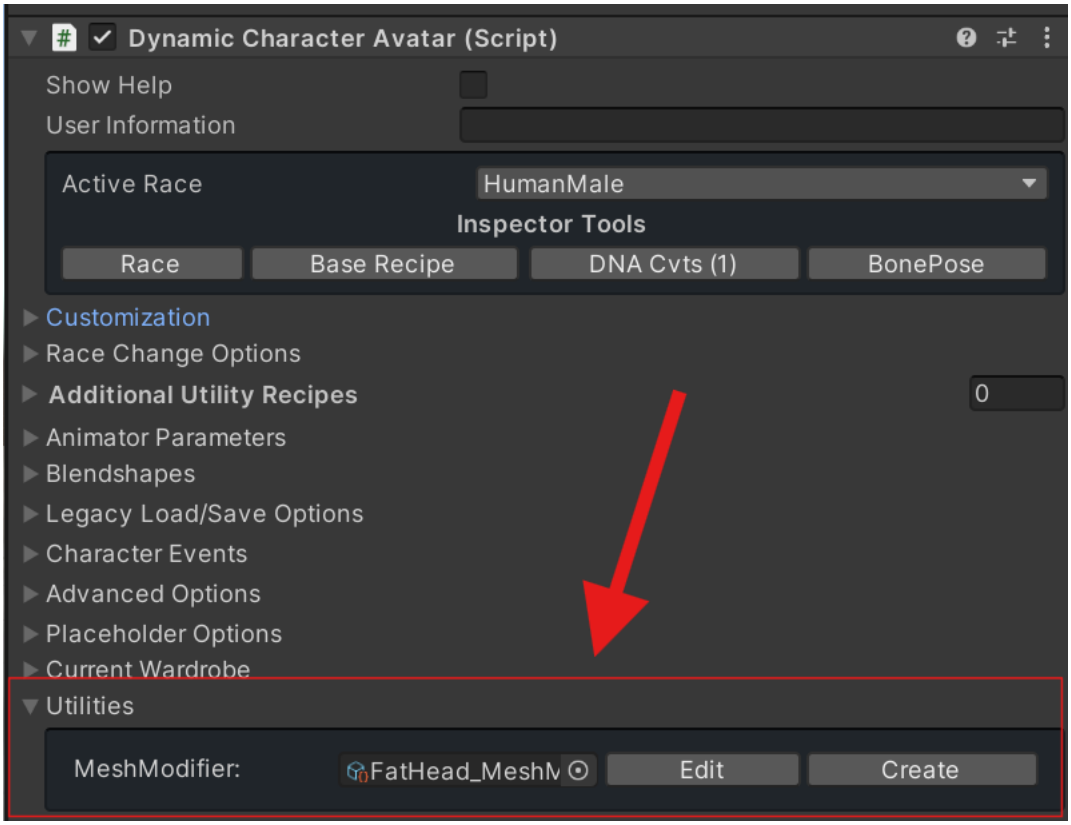
A Comprehensive Guide

What are Mesh Modifiers?

The Unity Multipurpose Avatar (UMA) system provides a powerful framework for creating highly customizable characters in Unity. Mesh Modifiers are an advanced feature that allows you to dynamically alter the mesh of your characters, adding a new level of customization and flexibility. This manual will guide you through the creation, editing, and application of Mesh Modifiers in UMA, ensuring you can leverage this feature to its fullest potential.

Accessing Mesh Modifiers

Unlike earlier versions, Mesh Modifier creation and editing are now accessed directly from the DynamicCharacterAvatar. This change allows you to see the effects of the modifiers in real-time while editing, providing a more intuitive and efficient workflow.



Prerequisites

Before you begin, ensure you have the following prerequisites:

- Unity installed (version 2022 or later recommended)
- UMA package installed in your Unity project
- Basic understanding of Unity's interface and UMA structure

Creating and Editing Mesh Modifiers

Step-by-Step Guide

To create and edit Mesh Modifiers, follow these steps:

1. Select the DynamicCharacterAvatar: In your Unity scene, select the DynamicCharacterAvatar component attached to your character. You should have “Editor Time Generation” enabled in order to edit and see the Mesh Modifiers in the editor. Ensure that you have the slots that you want to add modifiers for on the character!
2. Navigate to the “Utilities” Section: In the inspector window, find the Utilites section. Here you can create or edit Mesh Modifiers.
3. Create a New Mesh Modifier: Click on the "Create" button. This will open the editor, and allow you to define
4. Edit an existing Mesh Modifier: In the utilities section, drop (or select) the Mesh Modifier in the field provided. Click “Edit” to edit the Mesh Modifier in the editor.
5. Preview Changes: As you edit the Mesh Modifier, you can enable “Rebuild on change” on the Mesh Modifier window to observe the changes in real-time on your character model in the game view. This allows you to fine-tune the settings and ensure the modifications are applied correctly. Without this selected, you can rebuild the character at any time.
6. If some Wearable items are getting in the way, then from the Wearables window in the scene view, you can enable/disable them or even enable/disable them at the slot level. Note: Wearable is the new nomenclature for the upcoming changes to UMA. Wearable items will go into Wardrobe Areas.

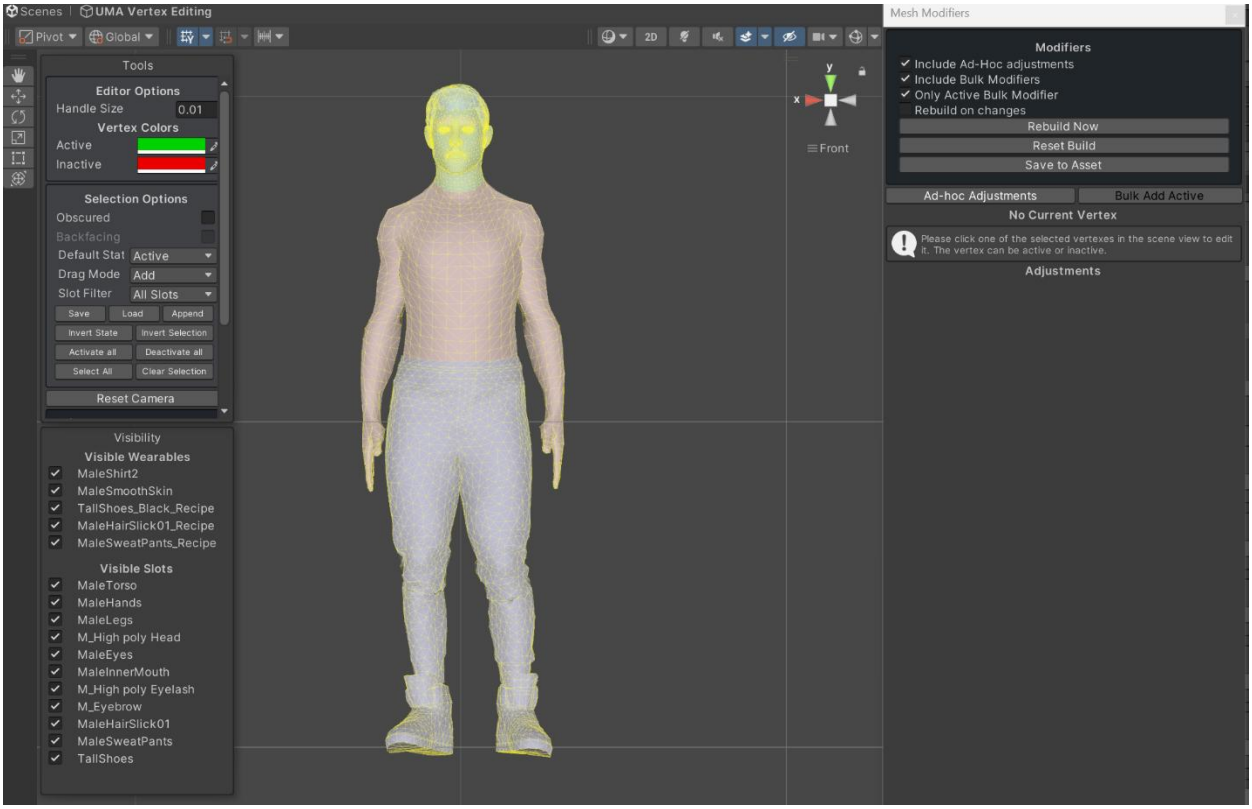
Using the Vertex Editor Tool

To start using the vertex editor, you must first select the desired vertices on your mesh. Selected vertices can either be Active or Inactive by default. To select a vertex, you can shift-click on it individually or use the box selection method to select multiple vertices at once. When dragging over vertices, you can add them to your selection, remove them from the selection, invert their selection state, or activate/deactivate the vertices.

Additionally, you can enable the selection of obscured or back facing vertices and limit your selection to a specific slot. Once you have made your selection, you can save or load the current vertex selection set, allowing for easy management of complex modifications.

Navigating through the scene follows the standard Unity format, using the alt key combined with mouse buttons. When selecting vertices individually, a shift-click will add to the selection or invert its state if it is already selected. To remove a vertex from the selection, simply control-click on it.

To reset the camera to the front of the character, use the “Reset Camera” button. Be aware that some options in the tools window might be hidden and may require scrolling to be seen. Note that only active vertices are added to bulk vertex modifiers.



Applying Mesh Modifiers in a Wardrobe Recipe

Adding Mesh Modifiers to a Wardrobe Recipe

Mesh Modifiers are typically added to a Wardrobe Recipe, which allows them to be applied dynamically based on the character's wardrobe. To add Mesh Modifiers to a Wardrobe Recipe, follow these steps:

1. Select the Wardrobe Recipe: In the UMA Wardrobe section, select the Wardrobe Recipe you wish to modify.
2. Navigate to the Modifiers Section: Locate the Modifiers section within the Wardrobe Recipe properties.
3. Add Mesh Modifiers: Click on "Add Modifier" and select the Mesh Modifiers you wish to include in the recipe. Configure the settings as needed.

Applying the Wardrobe Recipe

Once the Mesh Modifiers are added to a Wardrobe Recipe, they will be applied during the character build process if the character is wearing the recipe. This ensures that the mesh modifications are integrated seamlessly with the character's appearance.

Editing Mesh Modifiers Ad-Hoc

Real-Time Editing

One of the key advantages of Mesh Modifiers is the ability to edit them ad-hoc. This means you can make real-time adjustments to the modifiers without having to go through the entire build process.

Steps for Ad-Hoc Editing

1. Select the DynamicCharacterAvatar: In your scene, select the DynamicCharacterAvatar component of your character.
2. Navigate to the Mesh Modifiers Section: In the inspector window, locate the Mesh Modifiers section.
3. Edit the Modifier: Adhoc modifiers work on the “Current” vertex. To make a vertex current, make sure you have selected some vertexes, and then go into ad-hoc mode, and click on a vertex. It should begin flashing. Now you can add modifiers for that vertex, and edit values specifically.
4. Save Changes: Once satisfied with the modifications, save the changes to ensure they are applied during the next build process.

Editing Bulk Mesh Modifiers

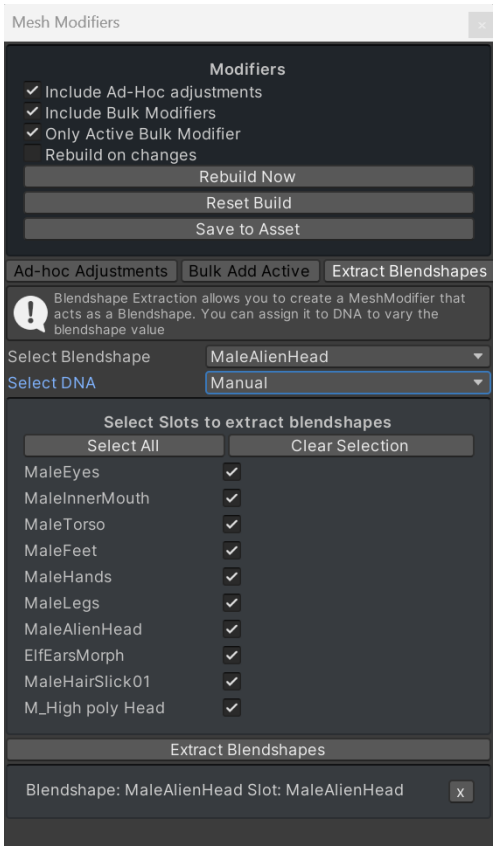
Steps for Bulk Editing

1. Build your character with the slots & wearable items that need to be present.
2. Edit or create a new modifier.
3. Select the Bulk Editing section from the top button bar on the Mesh Modifier window.
4. Select Modifier type: In the UMA Editor, select the Mesh Modifiers you wish to edit. Use the multi-select feature to choose multiple entries. When you have vertex selected and activated (green), select the Modifier type from the dropdown, and choose “Add Collection for selected vertexes”. A new collection of that type will be created, and you can edit all values at one time.
5. Edit Properties: Adjust the properties of the selected Mesh Modifiers. The changes will be applied to all selected entries.
6. Save Changes: Save your modifications to ensure they are incorporated into the character builds.

Extracting Blendshapes

Steps for Extracting Blendshapes

1. Build your character with slots & wearable items that need to be present.
2. Edit or create a new modifier.
3. Select the Extract Blendshapes section from the top button bar on the Mesh Modifier window.
4. Select the Blendshape you want to extract.
5. Select the DNA you want to use to apply the Blendshape (or leave “manual” to allow manual weight setting).
6. Select the slots you want to extract from (or press the Select All button).
7. Press the Extract Blendshapes button. Modifiers will be created for the “end frame” of the blendshape, for any slot that has that blendshape.
8. Save the changes to the asset.



Advanced Tips and Tricks

Optimizing Performance

When using Mesh Modifiers, it's important to consider performance. Here are some tips to optimize performance:

- Limit Complexity: Avoid using too many complex Mesh Modifiers on a single character to reduce processing overhead.
- Profile and Test: Regularly profile your project and test performance on various devices to ensure optimal performance.

Troubleshooting

Common Issues and Solutions

Here are some common issues you may encounter with Mesh Modifiers and their solutions:

- Modifier Not Applying: Ensure the Mesh Modifier is correctly added to the Wardrobe Recipe. Verify that the character is wearing the recipe during the build process.
- Unexpected Mesh Deformations: Double-check the settings of the Mesh Modifier. Small adjustments can sometimes lead to significant changes in the mesh. Note that all changes are to the *base mesh*, not to the transformed mesh.

- Performance Drops: Review the number and complexity of Mesh Modifiers applied. Optimize settings as needed and consider using LOD techniques.

Mesh Modifiers in UMA provide a powerful tool for customizing character meshes dynamically. By following this instruction manual, you can create, edit, and apply Mesh Modifiers effectively, enhancing the visual fidelity and uniqueness of your characters. Whether you are making ad-hoc changes or performing bulk edits, the flexibility and control offered by Mesh Modifiers will significantly enhance your character creation process in Unity.

Useful Links

Source Code

Stable Development Version
This branch will contain tested but possibly not release ready content.

Tutorials

Secret Anorak's UMA101
Secret Anorak's Content Creation
Secret Anorak's Race Creation
Casey's Dynamic Character Creation Tutorial
TheMessyCoder UMA Tutorial

Discord

Invite to Secret Anorak's Hideout

Content

Arteria3D UMA Section
Unity Asset Store
Github Base Content source

o3n Web Store o3n assets consist of a custom male and female race along with content compatible with those races. There are also some assets which are ported for the base UMA races. Please note that they cannot be used on base UMA races unless it is stated in the asset description.