Manual

Version 1.1.5

Overview

When characters wear clothing, you might see skin poking through during animation. This is most often seen with tight fitting clothing. A common solution is to manually create a new body mesh in modelling software and remove the parts of the skin that cannot be seen underneath the piece of clothing in question. Although time consuming, this solution will work for a character that can only wear a small amount of clothing. When the character is able to wear different **combinations** of clothing you will really start to get into trouble. When being able to wear e.g. a t-shirt, a jacket and a pair of gloves it will require 7 different body meshes and this number will grow exponentially as you add more. With Clothing Culler, you just configure the three pieces of clothing and it will sort out the combinations for you at runtime.

Features

- Endless combinations of clothing that cull the body and each other
- Runtime culling based on baked occlusion data
- Generate occlusion data
- Refine occlusion data by painting faces
- Low runtime overhead
- Easy to integrate
- Supports all render pipelines
- Supports LOD groups
- Supports blendshapes

In a nutshell

During the configuration process, you will reference prefabs so the Clothing Culler system knows which objects it has to work with. You will then configure occludee/occluder relationships between these prefabs so the system knows which prefab occludes which. Then you will generate and refine occlusion data for each relationship.

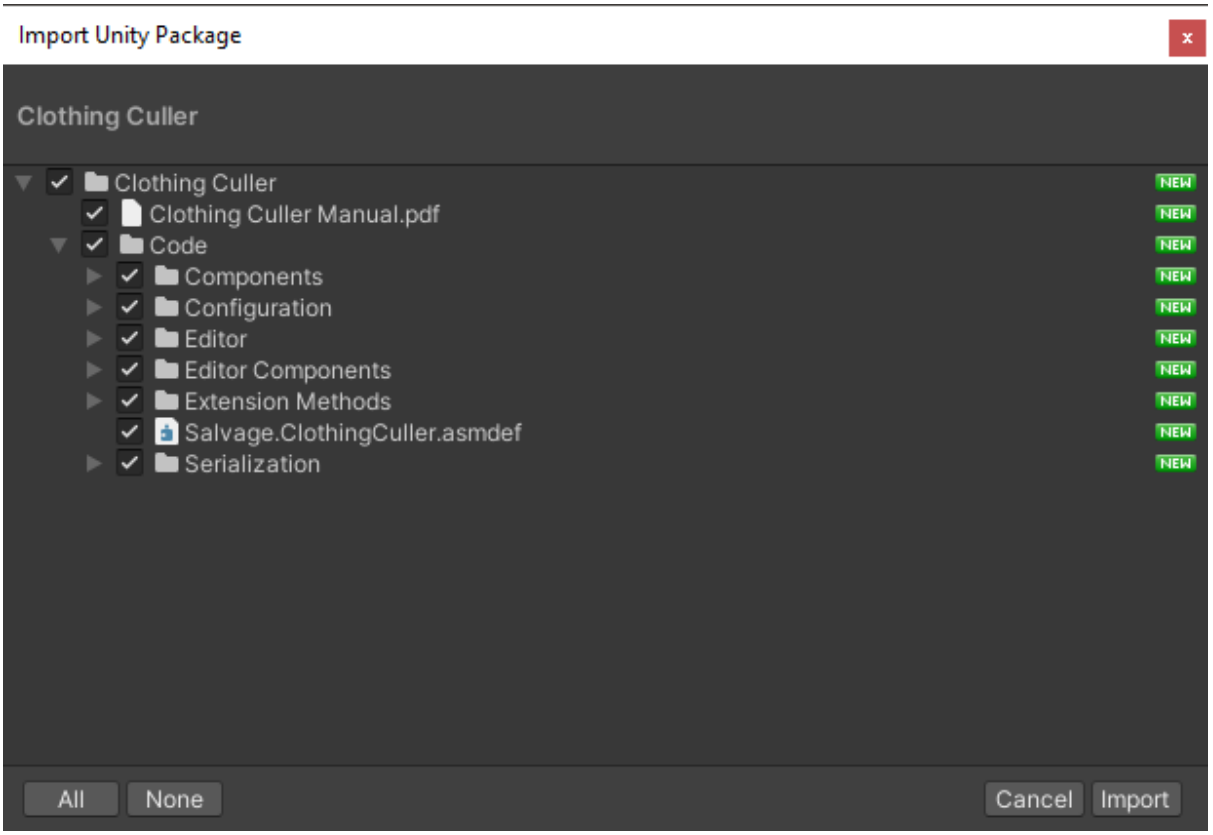
At runtime, you will register these prefabs at the Clothing Culler component. It will then figure out what needs to be culled, look up and combine the occlusion data and then use it to generate new partially-culled meshes.

Walkthrough

A walkthrough video is available here.

Table of contents

Package contents



These are the files you can expect when importing the Clothing Culler unity package. You can choose to move the Clothing Culler folder inside the plugins folder or anywhere else in your project.

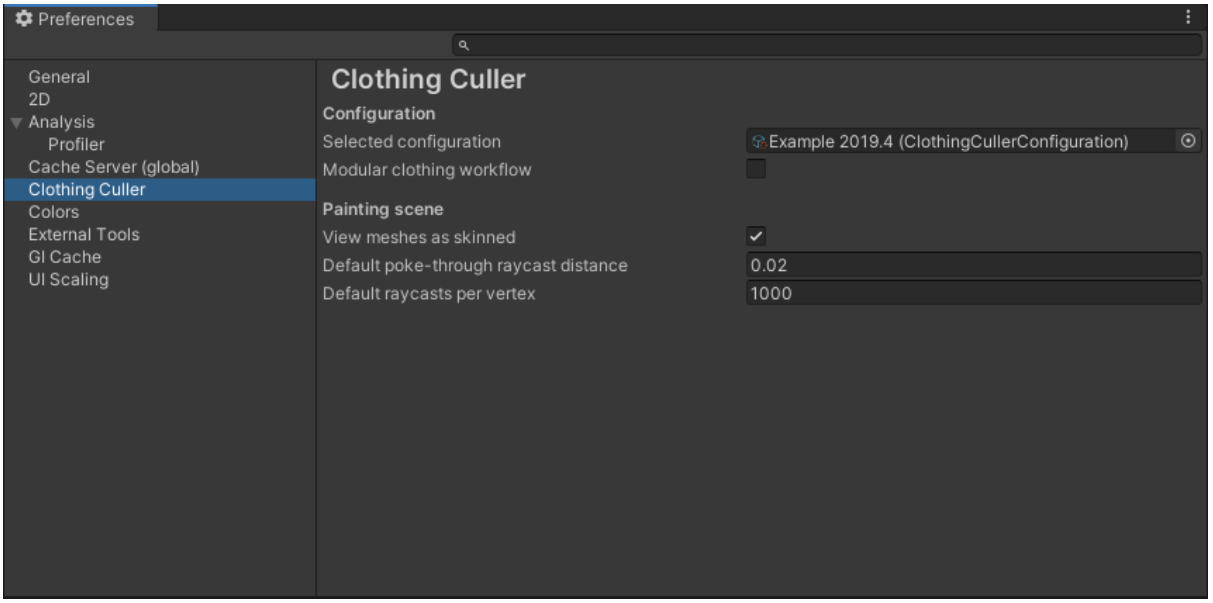
Editor windows

Clothing Culler comes with the following editor windows:

- **Resolve Skinned Mesh Bones** - optionally used to resolve bones when using the nested clothing workflow
- **Edit Configuration** - used to easily manage the system's configuration

Editor windows can be opened by navigating to **Window > Clothing Culler**.

Preferences



Name	Control type	Description
Selected configuration	Object field	Determines which configuration file will be edited in the *Edit Configuration* editor window.
Modular clothing workflow	Toggle field	Toggles the usage of modular clothing workflow on the selected configuration. When turned on, additional data will be generated inside every Occludee's config file which will be used at runtime.
View meshes as skinned	Toggle field	When turned on, meshes inside the View Occlusion Data scene will be using a SkinnedMeshRenderer. When turned off, a MeshRenderer and MeshFilter will be used.
Default poke-through raycast distance	Float field	The default *Poke-through raycast distance* value that will be used in the scene controls window.
Default raycasts per vertex	Int field	The default *Raycasts per vertex* value that will be used in the scene controls window.

Preferences can be opened by navigating to **Edit > Preferences**. Please note that this tab will not be visible until the *Edit Configuration* editor window has been opened for the first time.

Prefab setup

Before you can start the configuration process, you will need to set up prefabs. Prefabs require either a LODGroup component on the root, or a Renderer component somewhere in the hierarchy. When the system is using the prefab, it will look for a LODGroup component and if it is unable to find it, it will find all Renderer components in children and treat them as a single LOD.

Different workflows

Nested clothing

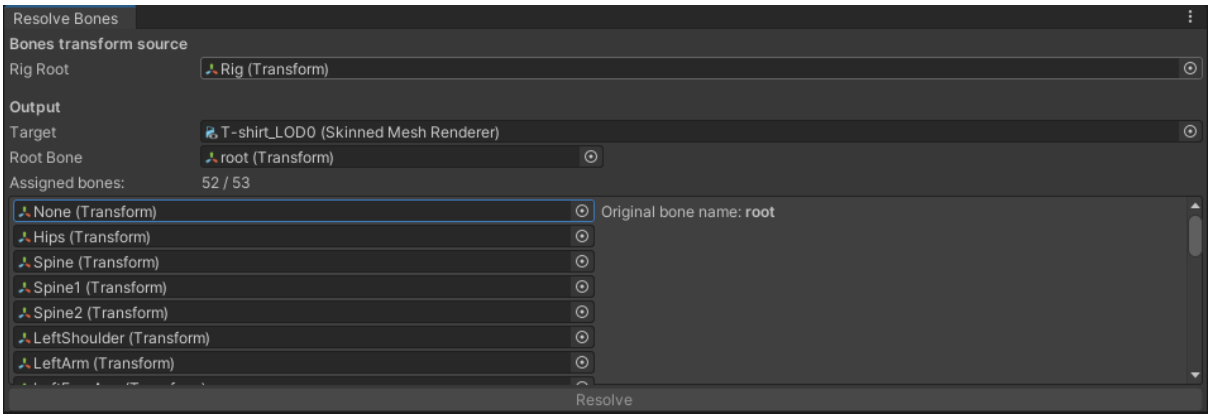
This workflow is the best choice for characters that can **not** change their clothing during the game, e.g. an A.I character. This means that all clothing that is part of the prefab only has to be registered at the Clothing Culler component once, and never deregistered.

The best way to achieve this is by creating a prefab for the A.I. character and creating a **nested prefab** for its body. Then you can compose the character's clothing by dragging clothing prefabs into it.

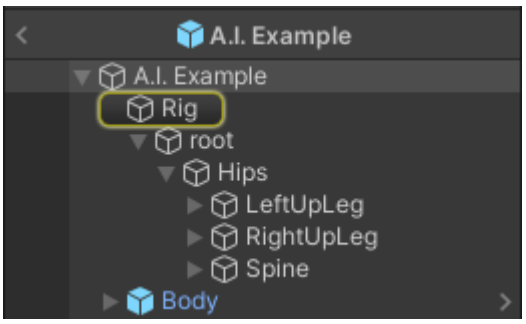
Resolve Skinned Mesh Bones

After you've added a clothing prefab to the A.I.Example's prefab, you will notice that it is invisible. This is because Unity can only serialize a SkinnedMeshRenderer's bones when the transforms are within the same prefab. In order to make it visible, you must assign the correct bones for each SkinnedMeshRenderer.

Clothing Culler comes with the *Resolve Skinned Mesh Bones* editor window that will guide you through this process. To open the window, navigate to **Windows > Clothing Culler > Resolve Skinned Mesh Bones**.



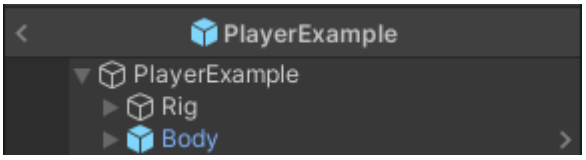
The first step is to assign the root transform of the A.I.Examples' rig as the *Bones transform source*. The second step is to assign the applicable SkinnedMeshRenderer as the *Output target*. The third step is to ensure all transform fields have been assigned to the correct bones. Lastly, use the *Resolve* button to update the SkinnedMeshRenderer's bones.



The first transform of the rig's bone hierarchy should be assigned to the *Rig Root* field.

Modular clothing

This workflow is the best choice for characters that **can** change their clothing during the game, e.g. the player's character. This means that when a piece of clothing is (un)equipped, it must be (de)registered at the player's Clothing Culler component.



The best way to achieve this is by creating a prefab for the player's character and creating a nested prefab for its body. Rather than adding all clothing prefabs to it, they should be instantiated / destroyed (or retrieved from / returned to an object pool) on demand.

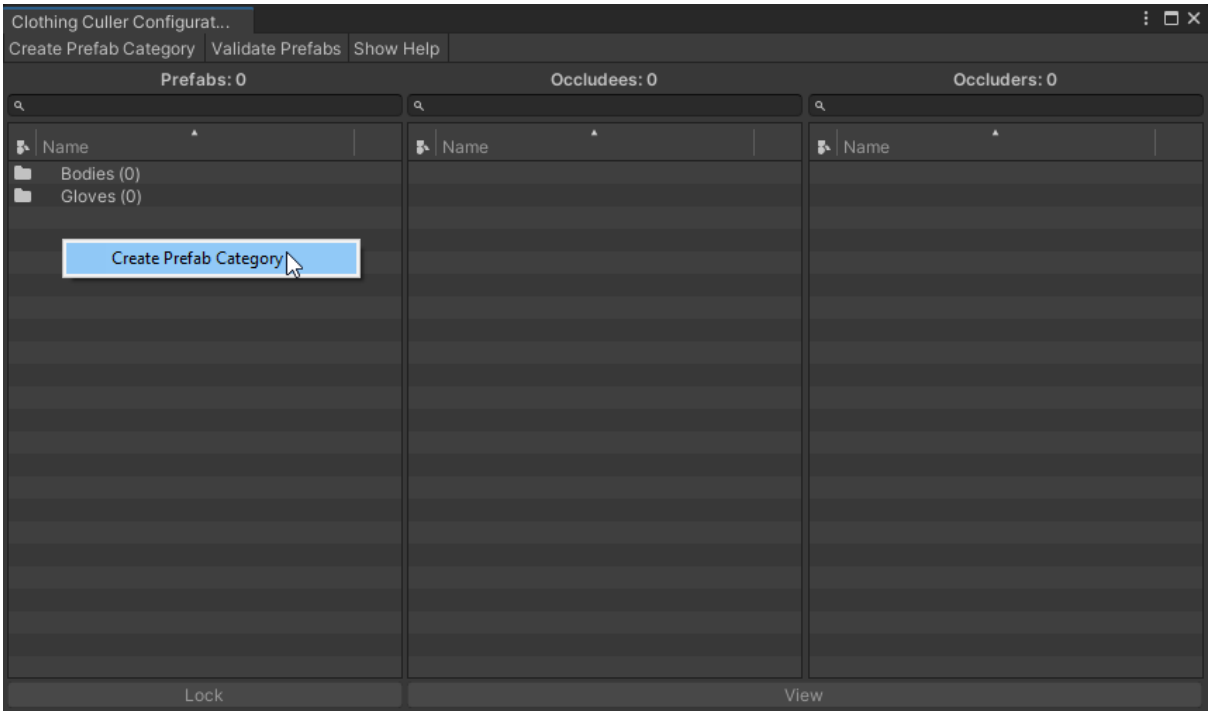
Enable the modular clothing workflow

In order to automate the process of resolving skinned mesh bones, the modular clothing workflow must be enabled in the preferences and on the PlayerExample's Clothing Culler component. When both settings are enabled, the system will automatically try to resolve the bones when an Occludee component is registered at the Clothing Culler component.

Configuration steps

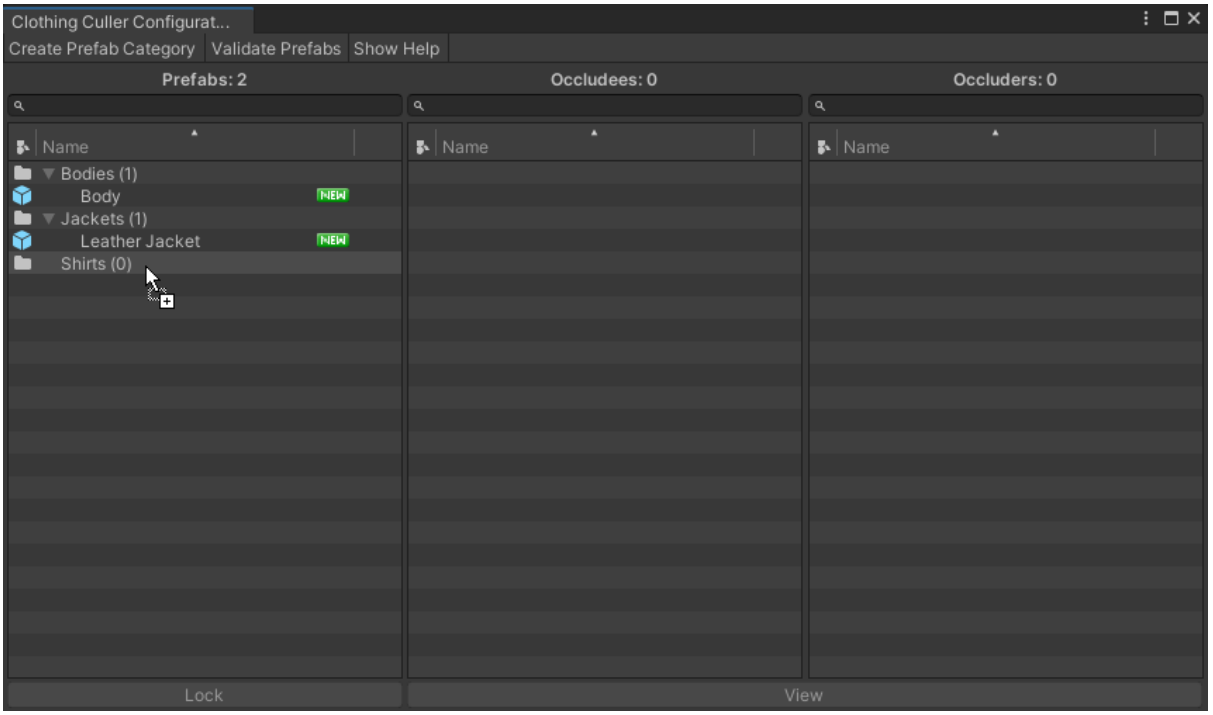
Most of the system's configuration is done inside the *Edit Configuration* editor window. To start the configuration process, open it by navigating to **Window > Clothing Culler > Edit Configuration**.

Create prefab categories



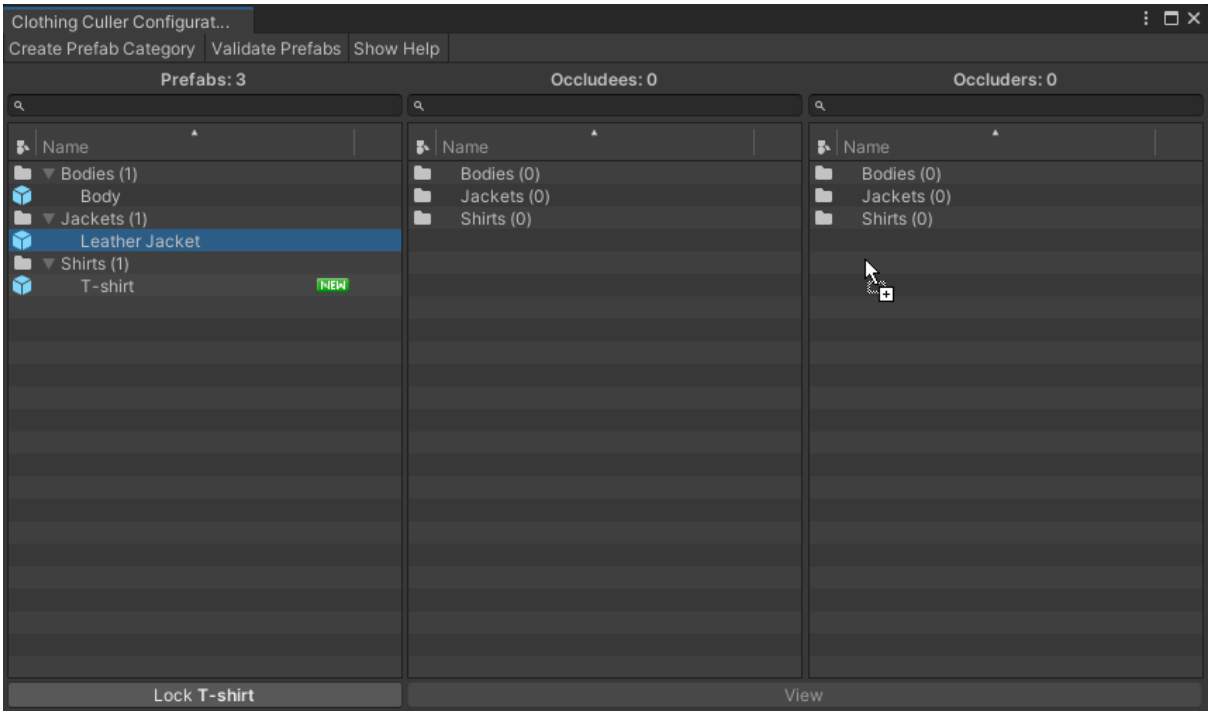
The first step is to create a prefab category. This can be done by using the *Create Prefab Category* button or from the *Prefab TreeView's* context menu.

Add prefabs

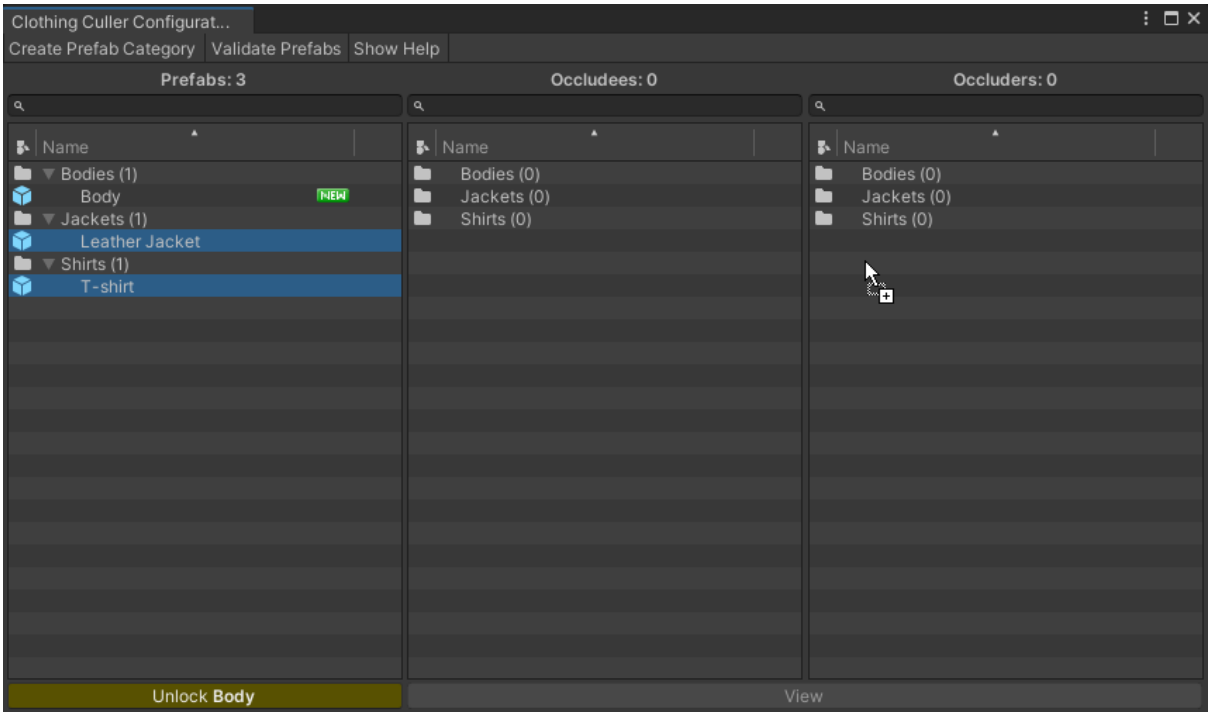


The second step is to add a prefab to a category by dragging it from the Project view into a category. This will also automatically add an Occludee component to the prefab, generate a configuration file for it and link them together.

Configure occludee/occluder relationships

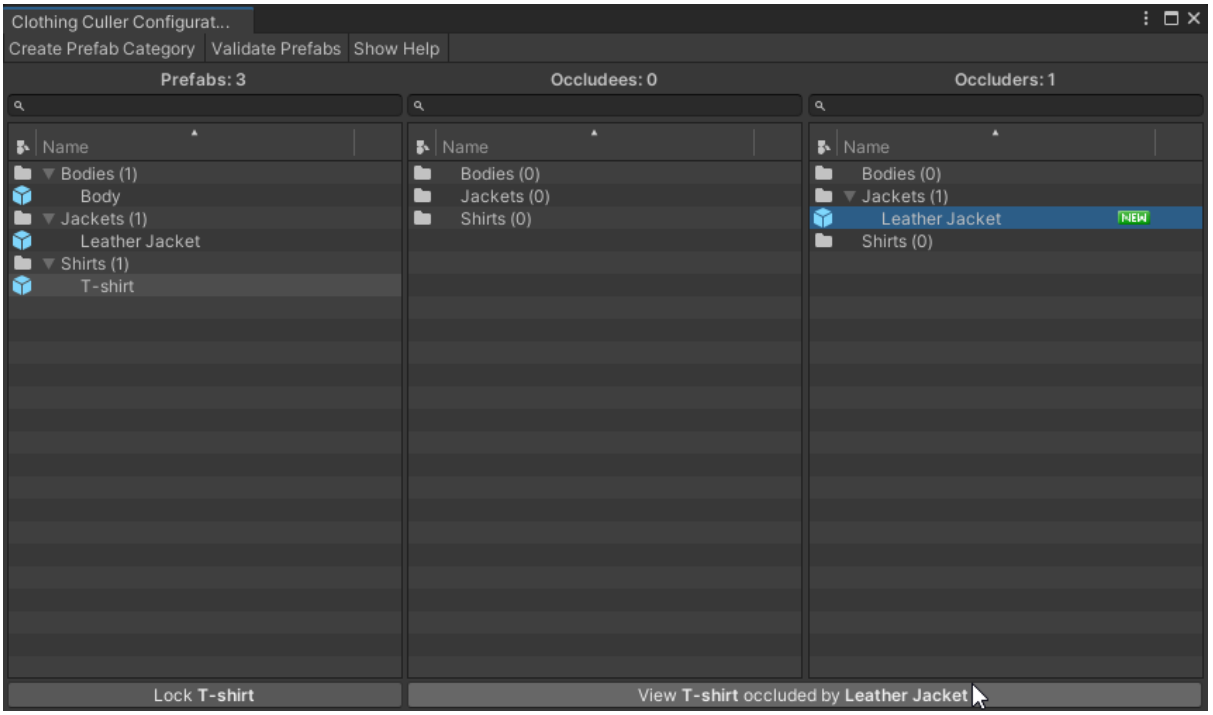


The third step is to configure occludee/occluder relationships between the newly added prefabs by selecting a prefab in the *Prefabs TreeView* and then drag another prefab or prefab category from the *Prefab TreeView* into the *Occludees or Occluders TreeView*.

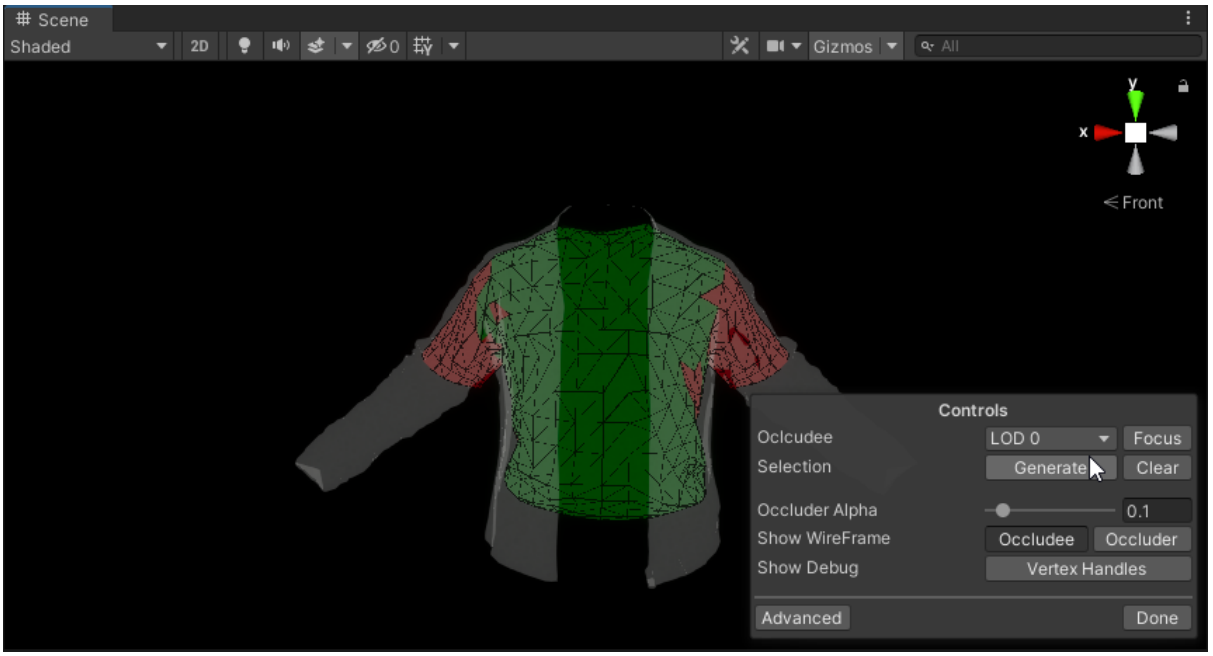


It's also possible to drag multiple prefabs or prefab categories at the same time by selecting a prefab in the *Prefabs TreeView* and then using the *Lock* button. You will then be able to drag the selected items into the *Occludees or Occluders TreeView* for the locked prefab.

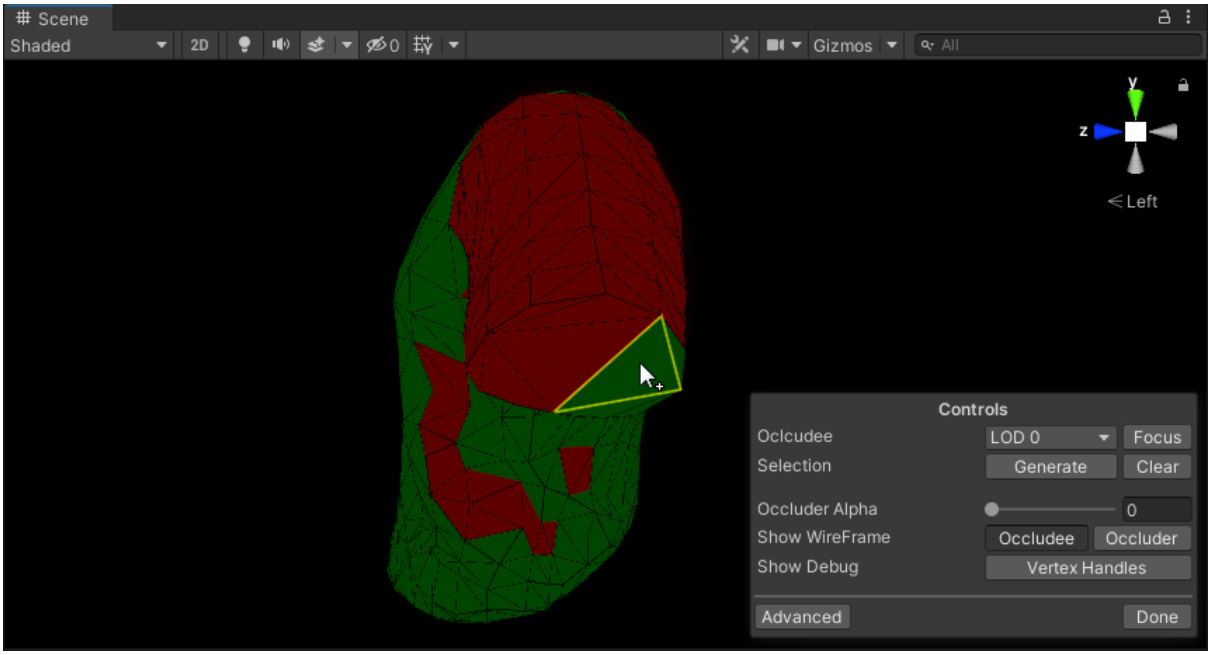
Generate occlusion data



The fourth step is to generate occlusion data for the newly added occludee/occluder relationships. This is done in a separate scene which can be opened from the View button.

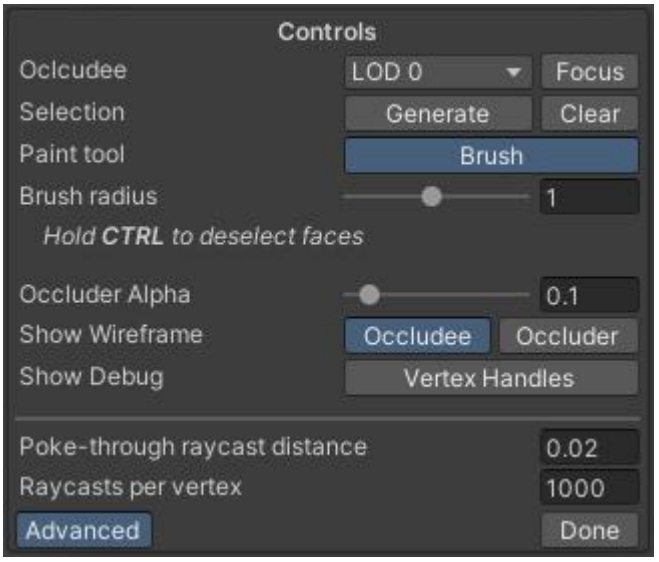


Once you are inside the scene, you will see both the Occludee and the Occluder's meshes. You can then generate occlusion data using the *Generate* button. Faces that are **green** indicate that they are visible and will not be culled. Faces that are **red** indicate that they are occluded and will be culled.



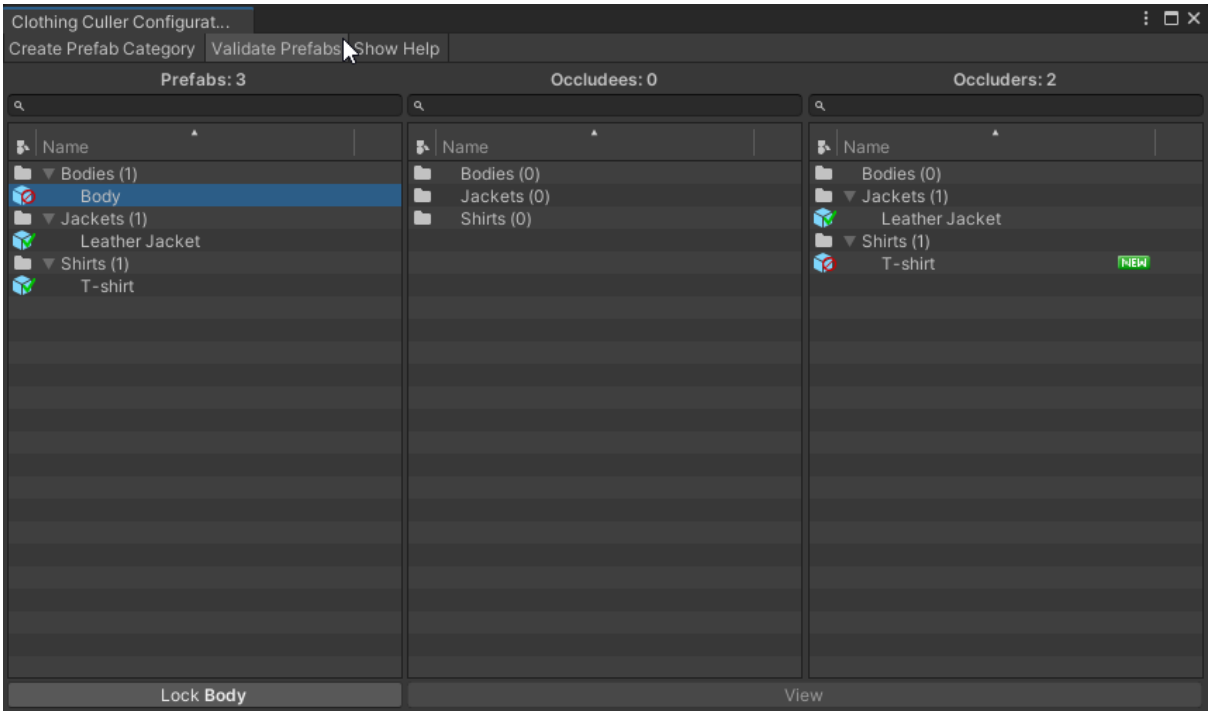
Although the auto generate will find most occluded faces, it's not perfect. If you need more fine control, you can paint faces using the cursor.

Scene controls



Name	Control type	Description
LOD x	Dropdown	View the occlusion data for the selected occludee's LOD.
Focus	Button	Focus the occludee's meshes in the scene view.
Generate	Button	Auto generates occlusion data for the occludee's meshes using raycasting from different angles.
Clear	Button	Clear the occlusion data for the occludee's meshes.
Brush	Button	Toggles Brush painting mode
Brush radius	Slider	Controls the radius of the painting brush
Occluder Alpha	Slider	Controls the transparency of the occluder's meshes.
Show Wireframe Occludee	Toggle button	Toggles the wireframe of the occludee's meshes.
Show Wireframe Occluder	Toggle button	Toggles the wireframe of the occluder's meshes.
Vertex Handles	Toggle button	Toggles the vertex handles debug mode and display of the *Show raycasts* field. This mode allows you to debug the auto generation by selecting a face and then choosing between three vertices to draw raycasts for.
Show raycasts	Mask field	Controls which type of raycasts to draw in the scene view.
Poke-through raycast distance	Float field	Controls the distance of the raycast that's fired to determine if the occludee's vertex is poking through the occluder's mesh.
Raycasts per vertex	Int field	Controls the amount of raycasts that are fired for each occludee's vertex to determine if the vertex is occluded by the occluder's mesh.
Advanced	Toggle button	Toggles the display of the *Poke-through raycast distance* and *Raycasts per vertex* fields.
Done	Button	Saves all made changes to disk and loads the previously opened scene.

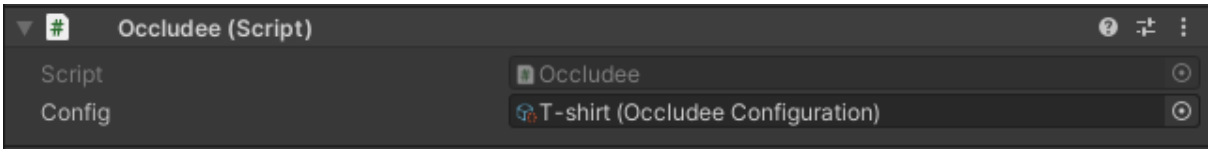
Validate configured prefabs



The final step is validating your configured prefabs by toggling the *Validate Prefabs* button. While the button is toggled on, the validator will run every time the window is focused. A **green** check mark indicates that the prefab is valid. A **red** red cross indicates that the prefab is invalid. A prefab is invalid when it doesn't have occlusion data generated for all occluders. In this example we can see that the T-shirt is a new occluder and the Body hasn't generated any occlusion data for it yet.

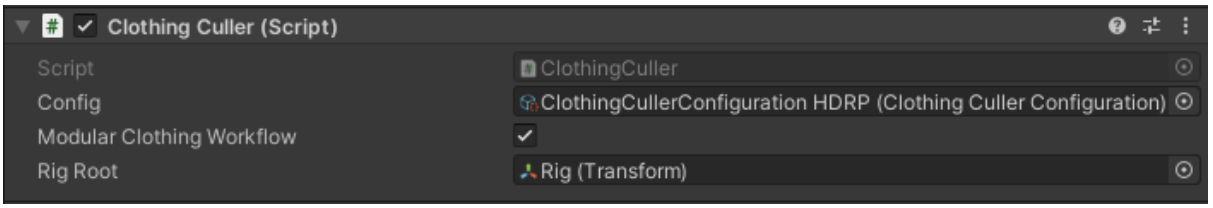
Scripting

Occludee

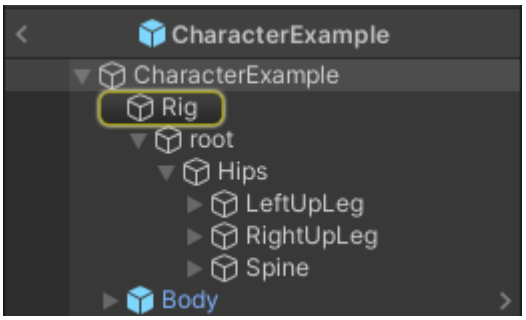


This component is automatically added to prefabs during one of the configuration steps. It has a reference to its configuration file, which holds all the occlusion data that will be used at runtime. This component should **not** be added or called manually.

Clothing Culler



This component is the heart of the runtime operations and must be manually added to your character prefab(s). It has a reference to the global configuration file, whether or not the character wants to use the modular clothing workflow and the rig root transform of the character prefab.



The first transform of the rig's bone hierarchy should be assigned to the *Rig Root* field and is only required when using the modular clothing workflow.

The main responsibility of this component is to keep track of all previously registered Occludee components, allow (de)registration of Occludee components and make them (un)cull each other when applicable.

The component has only two public methods:

- **Register** - register a new Occludee component and cull all previously registered Occludees when applicable
- **Deregister** - deregister a previously registered Occludee component and uncull all previously registered Occludees when applicable

If you are using the nested clothing workflow you can use the ‘RegisterAllOccludees’ script from the examples folder.

If you are using the modular clothing workflow you should create a script that registers the Occludee component of the instantiated body or piece of clothing to the Clothing Culler component of the character.

Please note that it’s also possible to exclude an Occludee from the modular workflow by passing in false for the *isModular* parameter of the Register method. This is particularly useful when the body of your character is already part of your character prefab, but the clothing is not.

Support

We are a small independent studio that greatly appreciates your feedback and support. If you have any questions, comments or feature suggestions don’t hesitate to reach out to us.

Join the Salvage Studios discord

The best way to get in contact is by joining our discord server.

E-mail us

If you like the good old way of emailing, we are happy to help you at support@salvage-studios.com.