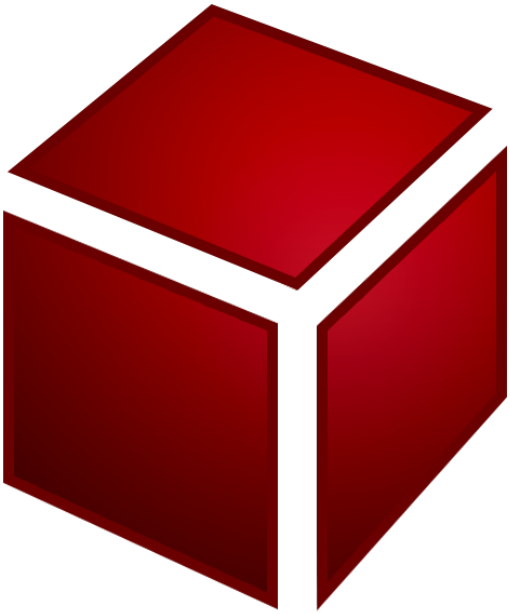
MeshKit - For Unity.

www.unitygamesdevelopment.co.uk

Created By Melli Georgiou



MESHKIT

Table of Contents

Version History

PLEASE BACKUP YOUR PROJECTS BEFORE UPGRADING.

v3.0

- You can now combine SkinnedMeshRenderers with two options (including one with texture atlasing!)
- More options for combining MeshRenderers with 32-bit meshes.
- You can now Separate SkinnedMeshRenderers.
- Changed "SeperateChildrenAtRuntime" component to "SeparateChildrenAtRuntime".
- Bug fixes and performance improvements.

v2.1.1

- Small bugfixes.

v2.1

- Updated for Unity 2019.3
- AutoLOD now allows you to assign the GameObject tag and layer per level.

v2.0.5

- Improved handling when combining certain meshes that exceed the maximum vertex limit.

v2.0.4

- Updated for Unity 2018.4

v2.0.3

- Updated for Unity 2018.3

v2.0.2

- Small fix for “Combine Children” runtime component when dynamically adding colliders.

v2.0.1

- Patch to fix a GUI display problem that some users were experiencing.

v2.0

There have been some API changes so please revisit the scripting section of this guide to find out more.

- Includes new Decimator and Auto LOD features that support both Mesh Filters and Skinned Mesh Renderers.
- Invert, Two-Sided, and Rebuild tool now support skinned mesh renderers (without sub-meshes).
- Rebuild tool expanded with new stripping options
- Editor tools now allow you to select Mesh Filters and / or Skinned Mesh Renderers.
- New Run-time components and scripting API for Decimator and Auto LOD tools.
- MeshKit Preferences allow you to choose between small and large icons in the MeshKit window.

v1.3

- Tested with Unity 2018.x
- The automatic Scene Fixer tool now updates relevant Mesh Colliders when scenes are duplicated.
- A help box has been added to the recreate normals tool.
- Bug fix when trying to create tangents on a Mesh without UVs.

v1.2

- Updated for Unity 2017.4
- Fixed harmless errors relating to GUI Layout in Unity 2017.x.
- You can now use threshold angles when calculating normals both at runtime and in the Editor.

v1.1.1

- Updated for Unity 5.5

v1.1

- Mesh Asset Management Updated For Unity 5.4
- You can now set the maximum number of vertices on each combined object.
- Light map UVs can be recreated in the Rebuild tab.
- [BUGFIX] Separating Meshes should now correctly remember the light probe settings in Unity 5.4 and higher.
- [BUGFIX] Fixed an issue where a mesh could have more than Unity's uppermost limit for vertices.

v1.0

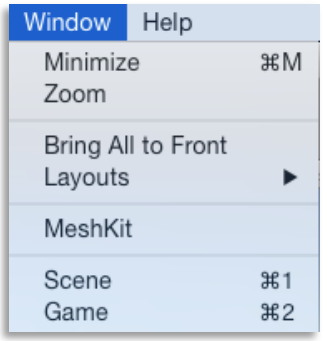
- First Commercial version of plugin.

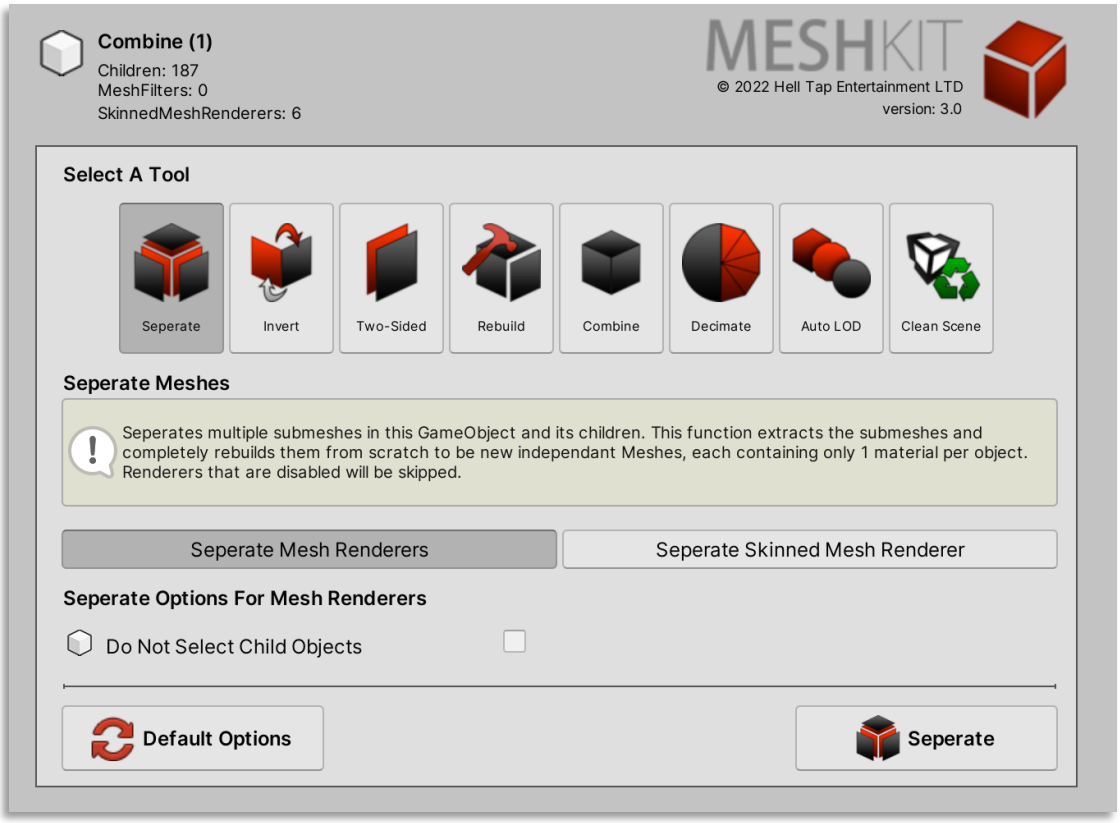
Installing and Accessing MeshKit

1) Install the package file into your project.

2) To open the MeshKit Editor, go to the Window menu and select “MeshKit”.

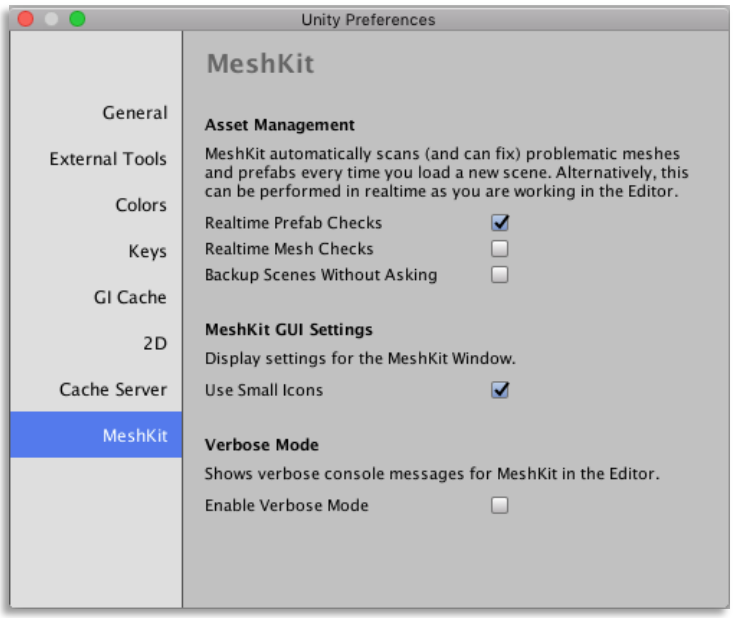
   You can then dock this window anywhere in the Editor, or keep it as a floating window.





MeshKit Preferences

MeshKit has a dedicated section within Unity's Preferences:



Asset Management

It is recommended to enable real-time Prefab and Mesh checks in the Asset Management section. This tracks your meshes while you make changes and helps prevent situations that may cause accidental deletion or modification (such as modifying a mesh being used in more than one scene, etc).

If these options are disabled, the checks are only performed upon loading a new Scene.

Automatic Scene Backups

Enabling this option removes the prompt for MeshKit to backup a scene before its first operation. It will backup your scenes automatically without asking you.

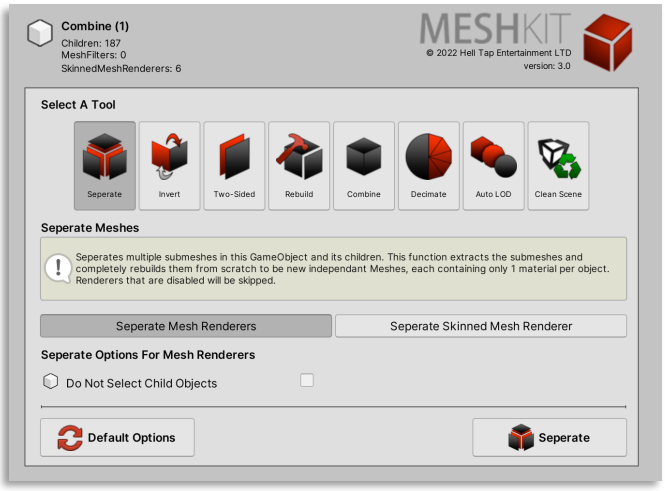
MeshKit GUI Settings

You can choose to select a single row of smaller icons or a larger set in the main MeshKit window.

Verbose Mode

Verbose Mode is an option primarily for us to figure out any issues or bugs that may be reported in future. It toggles a lot of information to be written to the console and generally shouldn't be needed while you're working in your projects.

The MeshKit Editor Tools



The MeshKit Editor is available when the scene has been saved and a valid GameObject is selected in the Hierarchy. It allows you to modify your meshes with a range of tools which can be selected from the top row of buttons:

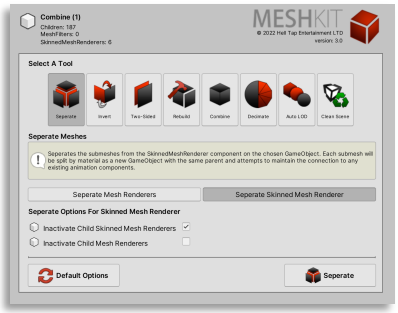
Separate Tool

The Separate tool comes in two workflows; one for Mesh Renderers and one for Skinned Mesh Renderers.

Separating Mesh Renderers

The Separate tool allows you to 'split' Mesh Renderers that have sub-meshes so that each sub-mesh becomes its own GameObject. This is very useful for combining meshes more effectively or further processing these meshes with the other tools. You can choose to separate the selected GameObject or the selected GameObject and all its children in the Hierarchy.

Separating a mesh effectively breaks the chosen object apart into new “separate” meshes so each part uses a single material. The original object ‘s MeshRenderer is “de-activated” in the Editor but the original object is kept in the scene just in case you are using a custom hierarchy or needed scripts. After separating, you can remove the original MeshFilter and MeshRenderer components from the object if you require.



Separating Skinned Mesh Renderers

The Separate workflow for Skinned Mesh Renderers is currently designed to work on a single object at a time.

SkinnedMeshRenderers are also separated by material into new GameObjects and meshes but the tool attempts to maintain the connection to any Animation / Animator components. However, please note that this tool cannot preserve any blendshapes after being seperated.

The tool also offers some 'clean up' options such as inactivating the GameObjects of any child SkinnedMeshRenderers (default = true) and MeshRenderers. This is helpful to quickly turn off the old Renderers so you can test your new setup before deleting them.

Invert Tool

The Invert tool flips a mesh's normals so it becomes inside out. This is useful in many cases. For example, using a house asset originally created to be looked at from the outside into an interior scene.

Two-Sided Tool

This is similar to the Invert tool but allows an object to be viewed from both sides. You should note that this increases the memory needed for the mesh.

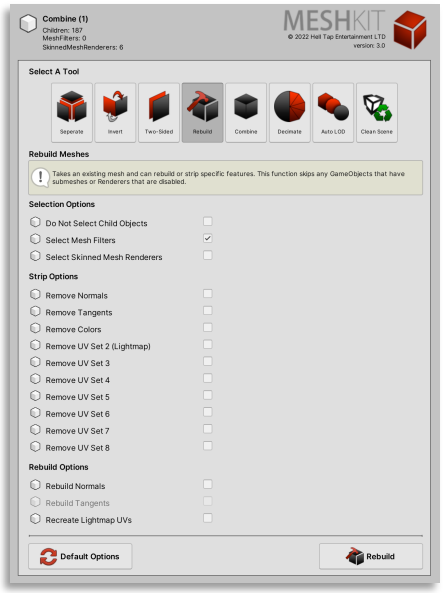
Another point is you should be careful when using this tool on objects that are already double-sided. While MeshKit does a great job of detecting its own Double-Sided meshes, this may result in doubling an already two-sided mesh.

Rebuild Tool

The Rebuild tool gives you the power to remove and recreate parts of a mesh on the fly without affecting the original asset.

The "Strip Options" allow you to remove the mesh data for normals, tangents, colors, and various UV sets. You will find this useful in removing data from meshes you don't need and freeing up memory.

The "Rebuild Options" offer a great way to add normals and tangents to meshes that need it, as well as creating a UV set for light mapping. This is especially useful after combining problematic meshes!



Combine Tool

The Combine tool is a fantastic way to optimize your scenes. It combines all the meshes in the selected GameObject and its children into the amount of materials being used.

For example, if you had 100 buildings using the same material, you could combine them to only use a single GameObject – this can dramatically increase the performance of your games.

The limitation here is the number of textures you use in the scene, the best results come with a good texture atlas – although MeshKit will squeeze the best

out of the meshes at hand. The Combine Tool comes in two workflows, one for MeshRenderers and one for SkinnedMeshRenderers as explained below:

Combining Mesh Renderers

When combining Mesh Renderers, the tool offers you the choice between allowing 32-bit meshes as well as the maximum number of vertices allowed per combined object.

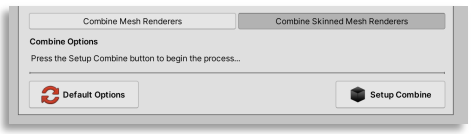
In short, 16-bit meshes are the most performant and compatible with all devices and platforms. The limitation is they can handle a maximum of 65,534 vertices per mesh.

Alternatively, you can choose to enable 32-bit meshes. This comes at a performance and bandwidth cost and is also incompatible with certain devices (such as the Mali-400 GPU on Android). The positive side is 32-bit meshes offer a theoretical maximum vertex count of about 4 billion. However, you are limited by the fact that Unity cannot serialize files larger than 4GB in size which on average would be about 20 – 25 million vertices.

NOTE: *If you are using extra UV channels, colors or other mesh data you should significantly reduce the max number of vertices to make sure the final meshes are not too large for Unity to save. In general, 5 – 10 million as a limit is usually a safe bet. In general, 16-bit meshes are recommended to maintain performance and compatibility across all devices.*

Combining Skinned Mesh Renderers

To combine Skinned Mesh Renderers, MeshKit requires a custom component be added to the selected GameObject. You can do this by pressing the 'Setup Combine' button.



You can undo any operations with the 'Uncombine' button or remove the component entirely by clicking the 'Remove Setup' button. The two combine modes for skinned meshes are described below:

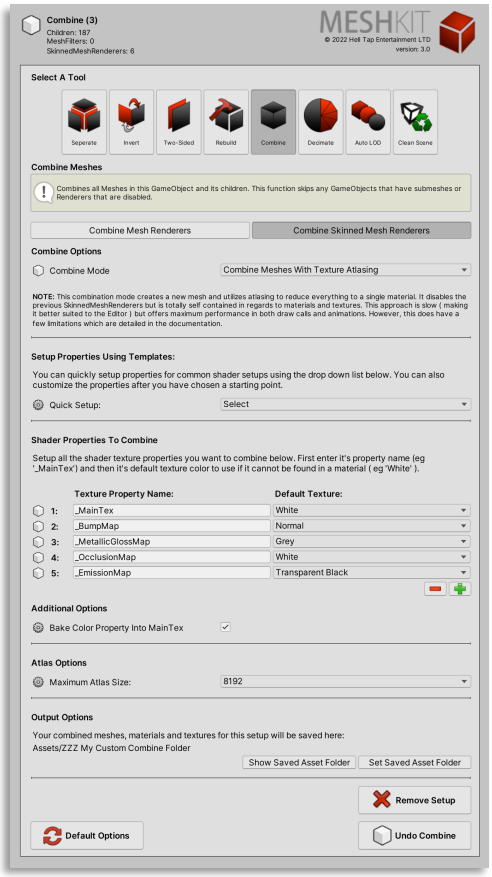
Combining Skinned Mesh Renderers *(With Material Arrays)*

The first combine mode creates a new mesh with a material array (sub-meshes). It disables the previous Skinned Mesh Renderers but requires the existing Animator / Animation components to run animations on the new mesh.

This approach is very fast (making it also a good choice for runtime) and compatible with almost any setup. Its primary advantage is performance improvements to animations and overhead.

Limitations and Notes: *This mode sets the 'AlwaysAnimate' culling option on the original animation components in order for it to work with the new Skinned Mesh Renderer. However, users can choose to optimize this further by creating a script to track the visibility of the new Renderer and toggle the animation component's culling type option accordingly.*

Combining Skinned Mesh Renderers *(With Texture Atlasing)*



The second workflow creates a new mesh and utilizes texture atlasing to combine and reduce everything down to a single material. It disables the previous Skinned Mesh Renderers but is totally self-contained in regard to materials and textures.

This approach is slow but offers maximum performance in both draw calls, animations and overhead.

Setup Properties Using Templates

In order to atlas the materials on the selected GameObject and its children, the tool needs to know what shader textures (properties) to combine. MeshKit provides a useful 'Quick Setup' drop down menu with some common presets for an 'Unlit', 'Bumped' and 'Standard' shader. This setup is presented in the 'Shader Properties To Combine' list which can be further customized if required.

Shader Properties To Combine

The Shader Properties list displays the name of the Texture Property on the left (e.g. _MainTex) and the default texture to use if one of one of the meshes does not have a texture setup for that property. In the example of the _MainTex property, a 'white' texture will be used by default if it is missing on other meshes.

This approach allows you to work with custom shaders with non-standard names and still combine your objects!

Additional Options

The tool also allows you to bake the _Color property of a shader directly into the _MainTex texture in order to preserve tints on specific objects.

Atlas Options

You can set a maximum atlas size for the tool to work with. 8192 offers the best quality but 4096 may offer a better compromise between memory and quality.

Output Options

Finally, you need to tell MeshKit where to save your assets. All of the new meshes and textures will be saved there. Because they are new source assets, MeshKit does this so they will not be considered 'managed meshes'.

Limitations and Notes

There are a few limitations to be aware of when using this workflow:

- *Firstly, as noted in the 'Material Array' workflow, this mode also sets the 'AlwaysAnimate' culling option on Animation components. However, the same workarounds can also be applied.*
- *Only the first UV channel of a mesh can currently be combined.*
- *Heightmaps / Parallax texture properties cannot be combined using this workflow.*
- *Materials with texture tiling and offsets are not currently supported and may produce strange results.*

If any of the above issues apply to your mesh, you can try using 'Combine With Material Arrays' instead.

Decimate Tool

The Decimate tool allows you to create lower-poly versions of your meshes (including Skinned Mesh Renderers) to drastically improve performance.

You can use the "Mesh Decimator Quality" slider to choose the target reduction of your meshes. A setting of 0 will result in maximum reduction and 1 preserves the quality of the mesh. For example, a setting of 0.5 will attempt to half the number of triangles in a mesh, 0.25 will attempt to reduce it by 75%.

Some meshes may prove to be problematic and result in artifacts such as gaps. To solve most of these issues, you can try using a mix of the "Preserve Borders", "Preserve Seams" and "Preserve UV Foldovers" options to stop the decimator removing important parts of the mesh (in exchange for less reduction).

Decimating works best on medium to high poly meshes and is a great way of improving performance by lessening the rendering overhead.

Auto LOD Tool

The Auto LOD tool is a powerful and easy way to setup LOD Groups within Unity. LOD Groups are a way to show lower poly versions of the same object as the camera moves further away from it. This can lead to significant performance increases without any visual loss in quality. This tool has a slight learning curve, as there are a lot of concepts involved.

Which GameObjects can be used with Auto LOD?

There are certain limitations on which GameObjects can be used with Auto LOD:

- Any GameObject with an attached Mesh Filter and / or children containing Mesh Filters.
- Any GameObject with an attached Skinned Mesh Renderer component.
- GameObjects with an LOD Group in a child or parent cannot be used.
- Skinned Mesh Renderers and Mesh Filters can't be combined together.
- Skinned Mesh Renderers can't have child objects with Skinned Mesh Renderers.

Setting Up The Auto LOD

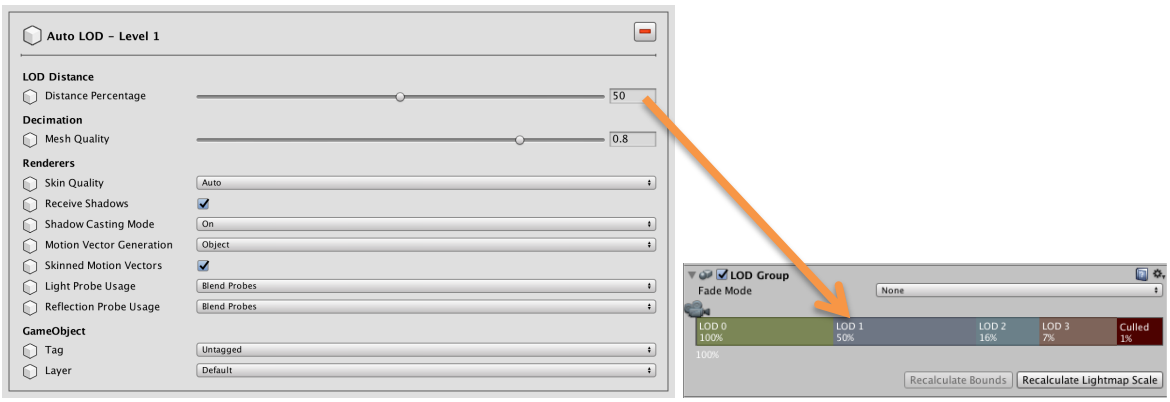
Once a valid GameObject is chosen, press the "Setup LOD" button for MeshKit to automatically create an LOD Group on the selected GameObject with 3 increasingly lower poly variations. A single button click is all that is needed!

Important Note: *After setting up the Auto LOD, the "MeshKit Auto LOD" Component is also added to the GameObject. This is used to keep track of your LOD Settings so changes can be made without having to start from scratch each time.*

Advanced Settings

If you would like manual control over the LOD Group, click the "Use Advanced Settings" checkbox.

Just like in the Decimator tool, you will notice Decimation options to preserve parts of the mesh. Under those options, you will find a list of LOD Settings that correspond to each of the LOD Levels in the LOD Group:



The "Distance Percentage" slider allows you to change the associated block of the LOD Group component's distance. This determines how far away the camera must be to show that LOD level. To make sure your settings are correct, MeshKit will force the distance percentage slider on each LOD Level to be smaller than the one before.

Important Note: *If you try to change the Distance Percentage on the LOD Group itself, MeshKit will override it based on your current settings. Make sure to update all settings in the MeshKit window!*

The next slider is "Mesh Quality" which controls how much decimation is applied. This works exactly the same as in the decimator tool. It makes sense to keep reducing Mesh Quality over each LOD Level.

The Renderers section reflects the settings of all Renderers with each LOD Level. For example, shadows are enabled on LOD Level 1, but disabled on 2 and 3.

You can also control the tag and layer on each LOD level.

If required, you can add and remove more LOD Levels to the LOD Group by clicking the "+" and "-" buttons respectively.

Finally, the culling distance slider is associated with the “Culled” block of the LOD Group. Changing this determines how far away the LOD Group must be before it is hidden.

LOD Tool Buttons

Changing the distance percentages are updated in real-time. However, if you make changes to decimation levels or renderers you must click the “Recreate LOD”. Doing this will remove the existing meshes and rebuild everything again based on your settings.

Clicking the “Reset LOD Setup” button will change the settings back to the defaults. However, you will still need to click the “Recreate LOD” button for the setup to actually be created.

To remove the LOD Setup completely, you can click the “Remove LOD” button.

Clean Scene Tool

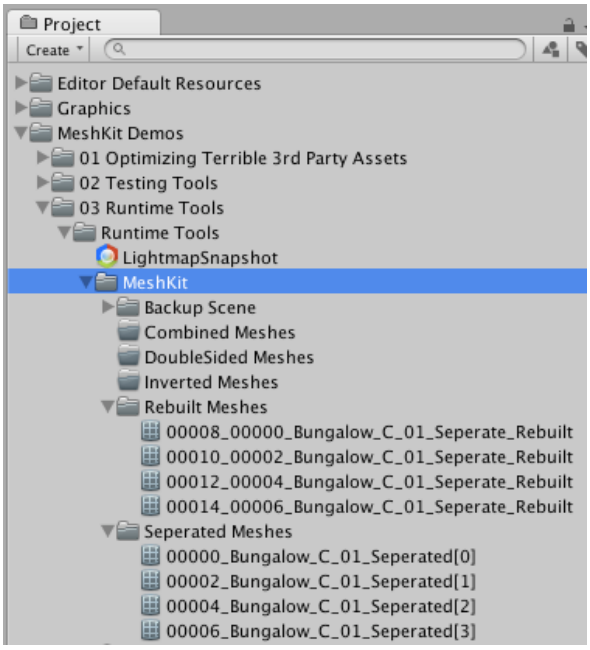
As you separate, modify and combine existing meshes, MeshKit creates new meshes in the background to reflect your changes. After a while, its likely there will be meshes that are no longer being used in the scene. The Clean Scene tool can automatically find and delete them for you!

Important Note: *Using this tool can break the tool’s UNDO step as it may delete meshes that were previously referenced.*

Understanding MeshKit's Asset Management

MeshKit is deeply embedded into the Unity workflow to make sure it doesn't get in your way. To do this, it employs its own mesh asset management system. It tracks your MeshKit meshes; keeps them organized and can delete unused mesh assets with a single click.

MeshKit creates a folder called "MeshKit" inside of the Unity Scene Folder (this is the folder with the same name as your scene - containing light map data and other scene related data).

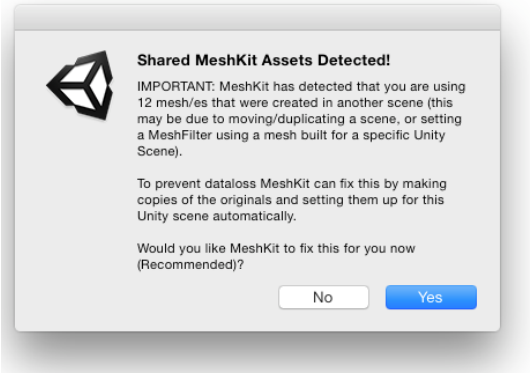


Inside this folder are the 'managed' local meshes that are created by MeshKit, as well as a backup scene that you may have opted to create before MeshKit's first operation in the scene.

This folder should NEVER be deleted unless you want to remove the entire scene and all of its local mesh assets.

It's also important not to move meshes outside of this folder as this is the area in which MeshKit will track its own meshes to prevent human-error related deletion of your meshes. Using the 'Clean Scene' tool will automatically remove unused Assets.

Duplicating Scenes Using MeshKit Meshes



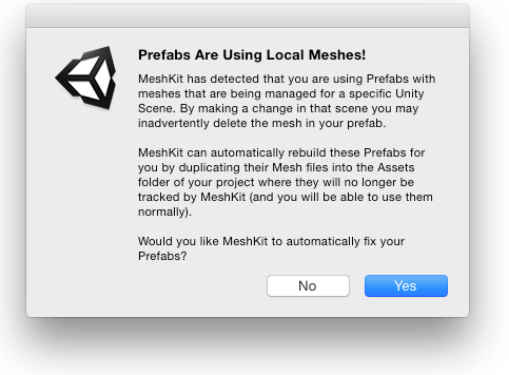
If you duplicate a Scene using MeshKit meshes (or try to use even a single Mesh from another MeshKit enabled scene), MeshKit will soon realize that you are using the same local meshes in 2 different scenes.

This causes a potential risk for data loss as if you make a change to the mesh or delete it in one of the scenes, it will affect the same mesh in the other scene – which is likely something you do not want to do!

MeshKit can automatically fix this by copying over the meshes and setting up all the affected MeshFilters in the new scene to use the updated references. This allows you to keep both scenes isolated and protected from any accidental mesh edits or deletions.

As of MeshKit 1.3, Mesh Colliders are also updated too!

Creating Prefabs With MeshKit Meshes

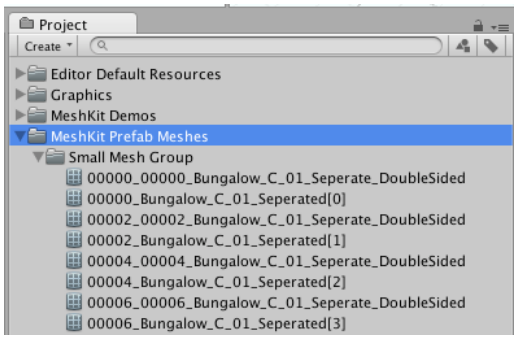


Of course, MeshKit also supports creating Prefabs out of the box!

As soon as you have created a prefab with MeshKit managed meshes (assuming prefab tracking is enabled) MeshKit lets you know that this could cause issues as you are trying to create a global object with local meshes.

MeshKit can automatically fix this for you by copying the local meshes out of the managed area into a new folder and updating the prefab's MeshFilter references to use the new meshes.

You will find the new directory in the root of the Project pane called "MeshKit Prefab Meshes". Each prefab found will be divided into subfolders with their associated meshes inside.



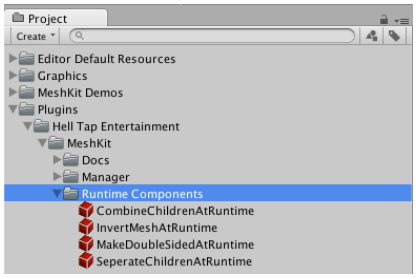
The Subfolders will actually use the same name as the Prefab itself, making everything easy to locate.

Though unlikely, if you create another prefab with the same name it is possible to overwrite some of the meshes and corrupt your prefab.

It is for this reason that we recommend you move the subfolder out of this directory as soon as it is created so MeshKit can no longer affect it.

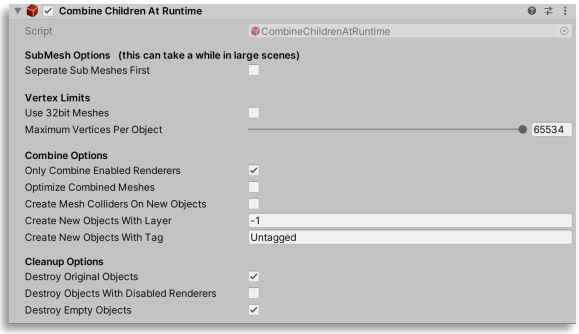
MeshKit Runtime Components

MeshKit offers runtime components of its main toolset found in the **Plugins > Hell Tap Entertainment > MeshKit > Runtime Components** folder.



Simply drag and drop them on a GameObject to use them!

Combine Children At Runtime



The Combine Children At Runtime component is a comprehensive tool.

The first thing to note is the "**Separate Sub Meshes First**" option in large scenes / objects can take a significant amount of time. During runtime this may give the appearance that things have crashed. For this reason, if you are using SubMesh Options it's probably better to use it during "loading" screens. Alternatively, separate the Meshes in the Editor beforehand if possible. You can uncheck "Strip Unused Vertices" in order to dramatically improve performance, but your meshes will not be optimized well for memory usage.

"Use 32-bit Meshes" allows you to enable support for huge meshes.

"Maximum Vertices Per Object" is a slider that can set the maximum number of vertices for every combined object. You can try lowering this number if you are having problems with larger meshes, but it will result in more objects being created. If you are using 32-bit meshes, the slider will be showing in "millions" so not keeping a limit of around 5 – 10 million is generally a good idea.

The "**Only Separate Enabled Renderers**" option allows you to choose whether to separate every child object or only those with their renderers enabled. The "**Only Combine Enabled Renderers**" works in the same way.

"**Optimize Combined Meshes**" allows you to run Unity's Optimize mesh function on the newly generated meshes as they are created – this increases the time it takes to combine objects.

"**Create Mesh Colliders On New Objects**" with automatically add a MeshCollider component on all new mesh objects that are created.

"**Create New Objects With Layer**" allows you to set a layer to assign all the new combined objects. -1 / 0 = default layer.

"**Create New Objects With Tag**" allows you to set a tag to all the new combined objects. "Untagged" is the default.

"**Destroy Original Objects**" will destroy the original meshes after combining.

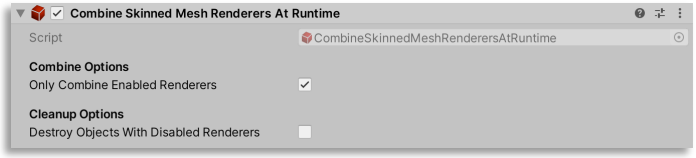
"**Destroy Objects With Disabled Renderers**" will scan through all the objects after combining and destroy any remaining object that has a disabled renderer.

"**Destroy Empty Objects**" will scan though all the objects after combining and destroy any GameObject that is empty and has no children.

Combine Skinned Mesh Renderers At Runtime

You component combines Skinned Mesh Renderers on this GameObject and its children.

This uses the "Material Array" approach discussed in previously as it is a fast algorithm suitable for runtime.



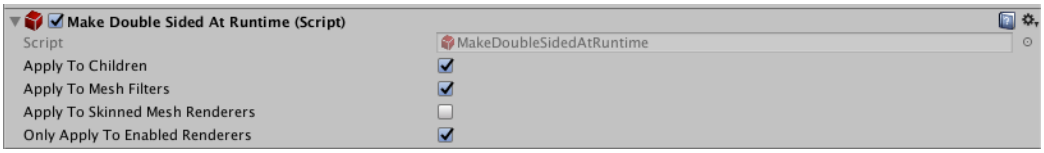
We can tell the component to only combine renderers that are enabled as well as the ability to destroy any GameObjects that are found to have disabled renderers when the process is complete.

Invert Mesh At Runtime

This component inverts a Mesh at runtime. It can also be applied to all child objects and limited only to enabled renderers.

Make Mesh Double Sided At Runtime

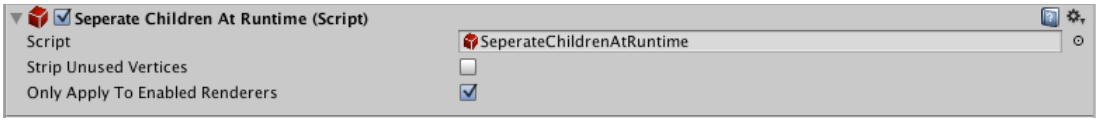
This component makes a Mesh two-sided at runtime. . It can also be applied to all child objects and limited to Mesh Filters, Skinned Mesh Renderers and enabled renderers.



Separate Children At Runtime

This component separates a Mesh and its children at runtime. It can also be limited only to enabled renderers.

Remember that separating large scenes / objects at runtime can take a lot of time, so it is generally advised to do this beforehand in the Editor if possible. You can de-activate “strip unused vertices” to make this much faster at runtime but your generated meshes will not be as optimized as doing it in the Editor. MeshKit allows you to make the choice between quality / performance.



Separate Skinned Mesh Renderers At Runtime

This component separates the attached Skinned Mesh Renderer at runtime.

There are some extra cleanup options available which take place after the process is complete:

"Destroy Original Skinned Mesh Renderer" will destroy the original component on the selected GameObject.

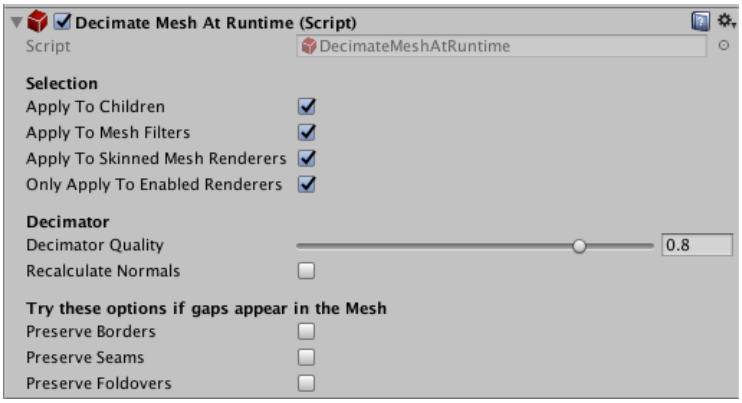
"Inactivate All Child Skinned Mesh Renderer GameObjects " will find all child Gameobjects that have SkinnedMeshRenderer components and inactivate them.

"Inactivate All Child Mesh Renderer GameObjects " will find all child Gameobjects that have MeshRenderer components and inactivate them.

"Destroy All Children With Disabled Renderers " will destroy the objects of the previous two options instead of inactivating their GameObjects.

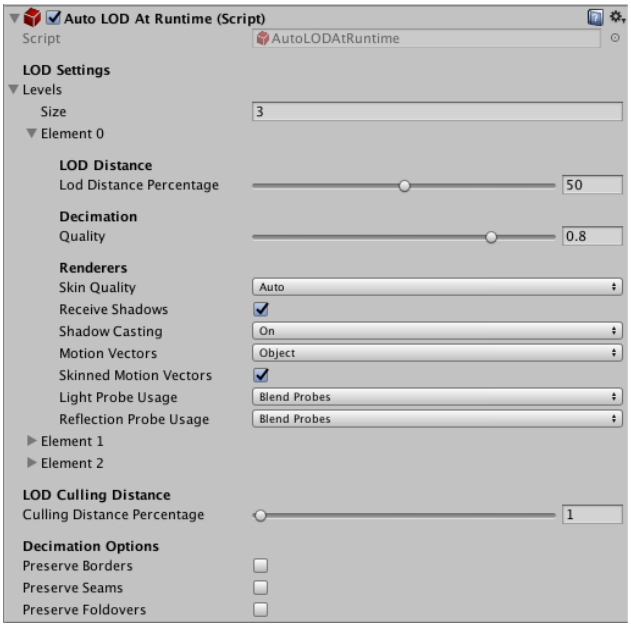
Decimate Mesh At Runtime

This component decimates a Mesh at runtime. It can also be applied to all child objects and limited to Mesh Filters, Skinned Mesh Renderers and enabled renderers. The options are identical to the MeshKit Editor Window.



Auto LOD At Runtime

This component attempts to create an Automatic LOD setup on the GameObject at runtime. The options are identical to the MeshKit Editor Window.



MeshKit API

MeshKit's tools are also available via the API. Remember to add "**using HellTap.MeshKit;"** in c# scripts or "**import HellTap.MeshKit;"** in Unityscripts. Skinned Meshes are not compatible with all methods.

Mesh Functions

```
MeshKit.RebuildMesh (
// Rebuilds a mesh by stripping and recreating features.
// NOTE: Using -1 for the rebuildNormalsAngle variable uses faster generation
// of normals at runtime. Otherwise, use 0 – 179 to apply a threshold angle.
// Extra UV channels will only be applied on Unity Versions that support it.
// Returns Mesh.
        Mesh m,
        bool optionStripNormals,
        bool optionStripTangents,
        bool optionStripColors,
        bool optionStripUV2,
        bool optionStripUV3,
        bool optionStripUV4,
        bool optionStripUV5,
        bool optionStripUV6,
        bool optionStripUV7,
        bool optionStripUV8,
        bool optionRebuildNormals,
        bool optionRebuildTangents
        float rebuildNormalsAngle = -1

);

MeshKit.StripUnusedVerticles (
// Strips a mesh, deleting unused vertices. Can also run Unity's Optimize function too.
// Returns Mesh.
        Mesh m,
        bool optimize
);

MeshKit.Strip (
// Strips a mesh, deleting unused data, with options to strip tangents, UVs, normals, etc.
// Returns Mesh.
        Mesh m,
        bool optimize,
        bool stripNormals,
        bool stripTangents,
        bool stripColors,
        bool stripUV,
```

```
        bool stripUV2,
        bool stripUV3,
        bool stripUV4,
        bool stripUV5,
        bool stripUV6,
        bool stripUV7,
        bool stripUV8,
        bool stripBoneWeights,
        bool stripBindPoses
);


MeshKit.SplitMesh (
// Splits A Mesh, and rebuilds each piece without any unused vertices.
// Returns Mesh[].
        Mesh m,
        bool stripUnusedVertices
);


MeshKit.CreateTangents (
// Recreates Tangents on an existing mesh
// Returns Mesh.
        Mesh m,
);

MeshKit.InvertMesh (
// Inverts an existing Mesh, flipping it inside-out
// Returns Mesh.
        Mesh m,
);

MeshKit.MakeDoubleSidedMesh (
// Rebuilds an existing Mesh, allowing it to be viewed from both directions.
// Returns Mesh.
        Mesh m,
);
```

Decimation Functions

MeshKit.DecimateMesh (
// Decimates the mesh of a Skinned Mesh Renderer using the quality setting: 0 = complete decimation, 1 = no decimation.
// Returns Mesh.
SkinnedMeshRenderer **smr**,
float **quality**,
bool **recalculateNormals**,
bool **preserveBorders** = false,
bool **preserveSeams** = false,
bool **preserveFoldovers** = false
);

MeshKit.DecimateMesh (
// Decimates the mesh of a Mesh Filter using the quality setting: 0 = complete decimation, 1 = no decimation.
// Returns Mesh.
MeshFilter **mf**,
float **quality**,
bool **recalculateNormals**,
bool **preserveBorders** = false,
bool **preserveSeams** = false,
bool **preserveFoldovers** = false
);

LOD Functions

MeshKit. AutoLOD (
// Automatically sets up an LOD Group (including decimated meshes) on a GameObject
// using default settings. Can be as simple as: MeshKit.AutoLOD(gameObject);
// Returns Mesh.
GameObject **go**,
bool **preserveBorders** = false,
bool **preserveSeams** = false,
bool **preserveFoldovers** = false
);

MeshKit. AutoLOD (
// Sets up an LOD Group (including decimated meshes) on a GameObject using
// AutoLODSettings[] and the culling distance.
// Returns Mesh.
GameObject **go**,
AutoLODSettings[] **levels**,
float **cullingDistance** = 1f,
bool **preserveBorders** = false,
bool **preserveSeams** = false,
bool **preserveFoldovers** = false
);

```
// Use the class below to setup AutoLOD levels . The fastest way to construct an
// AutoLODSettings variable is like this:
//  var lod = new MeshKit.AutoLODSettings ( lodDistancePercentageValue, qualityValue );

class AutoLODSettings {

        float lodDistancePercentage = 0.5f;
        float quality = 0.8f;
        SkinQuality skinQuality = SkinQuality.Auto;
        bool receiveShadows = true;
        ShadowCastingMode shadowCasting = ShadowCastingMode.On;
        MotionVectorGenerationMode motionVectors = MotionVectorGenerationMode.Object;
        bool skinnedMotionVectors = true;
        LightProbeUsage lightProbeUsage = LightProbeUsage.BlendProbes;
        ReflectionProbeUsage reflectionProbeUsage = ReflectionProbeUsage.BlendProbes;
        string tag = "Untagged";
        int layer = 0;
}

// ========================================
//          EXAMPLE CODE FOR SETTING UP LOD
// ========================================

// Setup LOD Levels
var lod1 = new MeshKit.AutoLODSettings ( 50f, 0.8f );
var lod2 = new MeshKit.AutoLODSettings ( 20f, 0.4f );
var lod3 = new MeshKit.AutoLODSettings ( 7f, 0.25f );
var levels = new MeshKit.AutoLODSettings[]{ lod1, lod2, lod3 };

// Setup Culling Distance
float cullingDistance = 1f;

// Perform the Auto LOD on the current gameObject:
MeshKit.AutoLOD( gameObject, levels, cullingDistance );
```

Batch Functions

MeshKit.Rebuild (

```
// Batch version of Rebuild function. Set recursive to true to rebuild children too.
// NOTE: Using -1 for the rebuildNormalsAngle variable uses faster generation
// of normals at runtime. Otherwise, use 0 – 179 to apply a threshold angle.
// Extra UV channels will only be applied on Unity Versions that support it.
// Returns void.
        GameObject go,
        bool recursive,
        bool optionUseMeshFilters,
        bool optionUseSkinnedMeshRenderers,
        bool optionStripNormals,
        bool optionStripTangents,
        bool optionStripColors,
        bool optionStripUV2,
        bool optionStripUV3,
        bool optionStripUV4,
        bool optionStripUV5,
        bool optionStripUV6,
        bool optionStripUV7,
        bool optionStripUV8,
```

```
        bool optionRebuildNormals,
        bool optionRebuildTangents
        float rebuildNormalsAngle = -1

);

MeshKit.InvertMesh (
// Batch version of Invert function. Set recursive to true to apply to children too.
// Returns void.
        GameObject go,
        bool recursive,
        bool optionUseMeshFilters,
        bool optionUseSkinnedMeshRenderers,
        bool enabledRenderersOnly
);


MeshKit.MakeDoubleSided (
// Batch version of MakeDoubleSided function. Set recursive to true to apply to children.
// Returns void.
        GameObject go,
        bool recursive,
        bool optionUseMeshFilters,
        bool optionUseSkinnedMeshRenderers,
        bool enabledRenderersOnly
);


MeshKit.SeparateMeshes (
// Separates the SubMeshes of a GameObject and its children (MeshRenderers).
// Strip unused vertices optimizes the meshes but can dramatically reduce performance.
// Returns void.
        GameObject go,
        bool enabledRenderersOnly,
        bool stripUnusedVertices
);


MeshKit. SeparateSkinnedMeshRenderer (
// Allows for runtime separating of Skinned Mesh Renderers.
// This method separates the SkinnedMeshRenderer on the current GameObject
// Returns void.
        GameObject go,
        bool destroyOriginalSkinnedMeshRenderer,
        bool inactivateChildSkinnedMeshRendererGameObjects,
        bool inactivateChildMeshRendererGameObjects,
        bool destroyAllChildrenWithDisabledRenderers
);
```

```
MeshKit.CombineChildren (
// Allows for runtime combining of MeshRenderers. Similar to runtime component.
// Returns void.
        GameObject go,
        bool optimizeMeshes,
        int createNewObjectsWithLayer,
        string createNewObjectsWithTag,
        bool enabledRenderersOnly,
        bool deleteSourceObjects,
        bool createNewObjectsWithMeshColliders,
        bool deleteObjectsWithDisabledRenderers,
        bool deleteEmptyObjects,
        int / uint userMaxVertices = maxVertices16 (65534)
);
```

```
MeshKit. CombineSkinnedMeshRenderers (
// Allows for runtime combining of Skinned Mesh Renderers.
// This method combines the objects with a new mesh and material array.
// Returns void.
        GameObject go,
        bool onlyCombineEnabledRenderers,
        bool destroyObjectsWithDisabledRenderers
);
```

Support

If you need any assistance or have suggestions for this plugin, feel free to visit our website at:

www.unitygamesdevelopment.co.uk

I hope you find this system useful, as I have in my own personal projects! =)

All the best!

- Mel