Runtime Mesh Batcher

Version 1.2.2

Contents

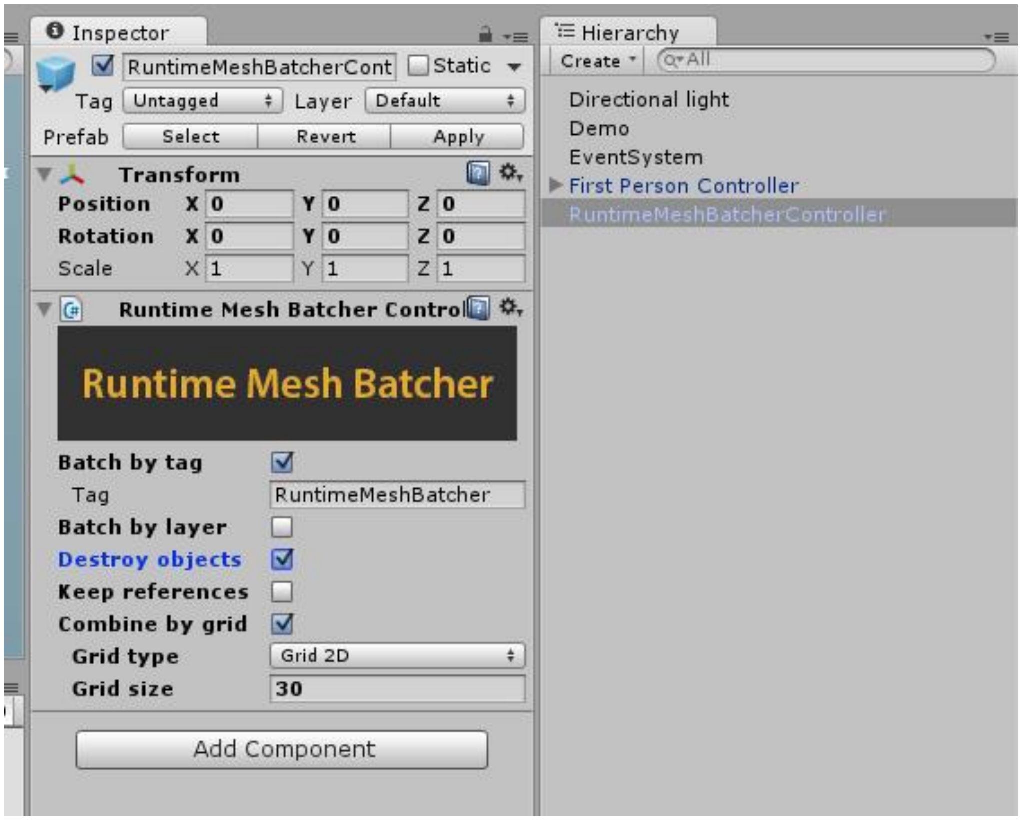
Installation **2**
Combining Meshes **2**
Option A: Automatic batching with the Inspector. 2
Option B: Automatic batching from script. 3
Option C: Batching with custom GameObjects array 4
Options **4**
Destroy original objects 4
Keep original object references 4
Batch By Grid 5

Installation

Copy the RuntimeMeshBatcher folder to your assets folder. If you are updating from a previous version, simply delete the previous version folder before installing the new one.

Combining Meshes

Option A: Automatic batching with the Inspector.



```
RuntimeMeshBatcherCont   Static
Tag Untagged   Layer Default
Position  X 0  Y 0  Z 0
Rotation  X 0  Y 0  Z 0
Scale     X 1  Y 1  Z 1
Batch by tag       ✓
  Tag              RuntimeMeshBatcher
Batch by layer     ☐
Destroy objects    ✓
Keep references    ☐
Combine by grid    ✓
  Grid type        Grid 2D
  Grid size        30
```

```
Directional light
Demo
EventSystem
▶ First Person Controller
RuntimeMeshBatcherController
```

One way to use the Runtime Mesh Batcher is to attach a controller script to a GameObject in the scene, and use this controller to configure options directly in the inspector.

1) Add the **RuntimeMeshBatcherController** prefab to the scene, or create an empty GameObject and attach the RuntimeMeshBatcherController script to it. *There can only be one RuntimeMeshBatcherController script in a scene.*

2) Configure the RMB Controller in the inspector window, specifying if you wish to **Batch by Tag** or **Batch by Layer**, and which Tag or Layer. Both options can be active at the same time.
3) Select if you wish to **destroy the original objects**, or **keep the original object references** (see options section below). Only one of those options can be active at the same time.
4) Select if you want to **Batch by Grid** and set up the options (see options section below).
5) Call `RuntimeMeshBatcherController.instance.CombineMeshes();` from one of your scripts to batch the meshes as per the controller settings OR if you prefer, check the **AutoRun** option in the inspector to execute the batching automatically at startup (useful if you wish to use RuntimeMeshBatcher on meshes created in the Editor, rather than instantiated at Runtime).

Option B: Automatic batching from script.

The other way is to directly call the batching system from code, by calling the function `RuntimeMeshBatcher.CombineMeshes`:

```
RuntimeMeshBatcher.CombineMeshes(
        [bool processObjectsByTag = false],
        [string rmbTag = null],
        [bool processObjectsByLayer = false],
        [int rmbLayer = 0],
        [bool destroyOriginalObjects = false],
        [bool keepOriginalObjectReferences = false],
        [bool combineByGrid = false],
        [MeshBatcherGridType gridType = MeshBatcherGridType.Grid2D],
        [float gridSize = 0]);.
```

All parameters above are optional, you only need to specify those for which you wish to change the default value.

So if you want to combine objects by tag, you might call:

```
RuntimeMeshBatcher.CombineMeshes(processObjectsByTag: true, rmbTag:
        “RuntimeMeshBatcher”);
```

Or more simply:

```
RuntimeMeshBatcher.CombineMeshes(true, “RuntimeMeshBatcher”);
```

Option C: Batching with custom GameObjects array

If you do not wish to use Tags or Layers and let Runtime Mesh Batcher automatically gather objects, you may also provide your own array of GameObjects, optionally providing additional options:

```
RuntimeMeshBatcher.CombineMeshes(
        GameObject[] gameobjects,
        [bool destroyOriginalObjects = false]
        [bool keepOriginalObjectReferences = false]
        [bool combineByGrid = false],
        [MeshBatcherGridType gridType = MeshBatcherGridType.Grid2D],
        [float gridSize = 0]);.
```

Since v 1.2.2, you can also use the inspector to set up options visually, and just pass a custom GameObjects array to the controller like this:

```
RuntimeMeshBatcherController.instance.CombineMeshes(GameObject[]
        gameobjects)
```

Options

By default, Runtime Mesh Batcher keeps all original meshes in place, and just turns off the renderers. The new combined meshes are placed as children to a parent object, incrementally named "RuntimeMeshBatcherParent_n". The `CombineMeshes` function returns a reference to this parent GameObject, which you can keep around if you wish to delete or deactivate the meshes, or reverse the process. These objects are in turn parented under the RuntimeMeshBatcherController GameObject, or a new created top-level container.

Destroy original objects

If you do not need the original GameObjects around, for example to keep active colliders on them, you may ask Runtime Mesh Batcher to destroy them. Be aware however that if it frees memory in the long term, destroying a large amount of objects has a noticeable performance impact on the frame it happens. You should only use this option when batching at level load time, not during gameplay.

Keep original object references

You can ask Runtime Mesh Batcher to keep a reference to the original GameObjects, allowing it to reverse the process. This option is false by default, to not waste memory if you do not require this functionality. If "Keep original object references" is checked, this enables the

`UncombineMeshes` method. To use it, you must provide the reference to the parent object returned from a previous `CombineMeshes` call, and as with combine meshes you can call it either through the controller with:

```
RuntimeMeshBatcherController.instance.UncombineMeshes(rmbParent);
```

or directly with a static method:

```
RuntimeMeshBatcher.UncombineMeshes(rmbParent);
```

This deletes all newly combined meshes and re-enables the MeshRenderers on the original objects. Note that the “Destroy original objects” and the “Keep original object references” options are mutually exclusive.

Batch By Grid

You can ask Runtime Mesh Batcher to batch objects together following a grid, making many smaller meshes instead of huge ones. This may result in a lesser draw calls reduction, but allow different culling procedures, such as frustum culling, distance culling or other custom culling scripts. Depending on your situation, this may provide a better performance. There are two options:

- **Grid type**: The batching can follow a 2d grid, along the ground plane (XZ axes) or a 3d grid (XYZ axes). If you are using the inspector controller, you can simply pick up the option you wish from the drop-down menu. If you are using Runtime Mesh Batcher from script, you have use the **`MeshBatcherGridType`** enum when passing the `gridType` parameter to the `CombineMeshes`() method, using one of those two values:
    - `gridType = MeshBatcherGridType.Grid2D`
    - `gridType = MeshBatcherGridType.Grid3D`
- **Grid size**: The grid spacing. The value is the same for all axes. The value (a **`float`**) must be higher than 0, or Runtime Mesh Batcher will ignore the Batch By Grid option.