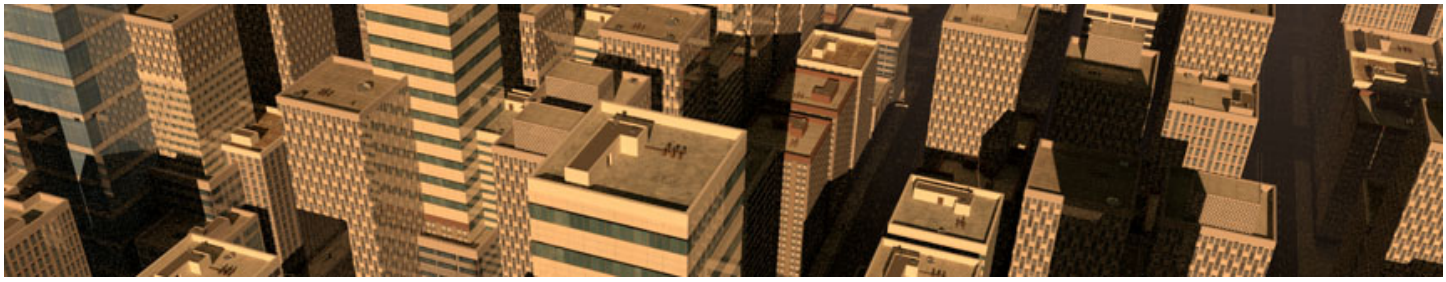
Digital Opus

MESHBAKER LOD MANUAL



Mesh Baker LOD organizes level of detail meshes into clusters. The clusters are baked into combined meshes at runtime. The combined meshes are updated as LOD objects move closer to and further from the camera.

Requires Mesh Baker 2 **– available in the Asset Store**

In order to understand how Mesh Baker LOD works it is essential to have at least a basic understanding of atlases and how Mesh Baker works. If you are new to Mesh Baker, then take some time to read the documentation, look at the YouTube videos and play with the examples.

Definitions

LOD Objects

An `LOD object` is a GameObject with one or more child game objects, each of which has a Renderer with a mesh at a different level of detail.:

- Game Object (MB2_LOD component goes here)
  - LOD0 Game Object (has at least one MeshRenderer or SkinnedMeshRenderer somewhere in its children)
  - LOD1 Game Object (has at least one MeshRenderer or SkinnedMeshRenderer somewhere in its children)

- LOD2 GameObject (has at least one MeshRenderer or SkinnedMeshRenderer somewhere in its children)

LODManager

Each scene will have a single `LODManager`. `LOD Objects` notify the `LOD manager` when there is a change in their level of detail. The `LOD manager` contains a list of `bakers` that do the actual baking. The `LOD manager` prioritizes, queues and schedules baking and assigns `LOD objects` to `bakers`.

Bakers

A `baker` contains a `Mesh Baker` and a list of `clusters`. Each `LOD object` is assigned to a `baker` based its materials, renderer, lightmap index etc. The `baker` in turn assigns the `LOD object` to a `cluster`, where it gets baked.

Clusters

The game world is divided into 3D volumes or cells called `clusters`. Each `LOD object` in the scene is mapped to the `cluster` which contains its origin. Each `cluster` is baked into one or more combined meshes. An `LOD object` must always belong to the same `cluster` (which is why movable objects must use the simple clustering scheme).

There are three clustering schemes:

- **Simple**: All meshes assigned to this `baker` are combined into one big global combined mesh. This scheme is used for non-static and skinned meshes. The meshes contained in the combined mesh can be distributed through the scene which may not be efficient if the meshes are far apart.
- **Moving**: Moving works like Simple except it has bounds which are updated when the cluster's LODs are checked. It works well with moving meshes that are grouped together such as a unit of soldiers. You may need to use labels to map all members of a unit to the same baker if there is more than one unit in the scene.
- **Grid:** Space is divided into cubic volumes and all meshes in each volume are combined.

Instructions

Overview Of The Workflow

The recommended workflow is to do the following in a setup scene free from the clutter of your game level.

- Group Objects And Bake Combined Material Assets
- Add, Configure And Prefab the LOD Manager
- Add LODCamera Components To Your Cameras
- Create LOD Prefabs

Once this has been done it is easy to use and re-use the setup in your game level scenes by dragging the prefabs into your scenes and adding MB2_LODCamera components to your cameras.

Step 1 Group Objects And Bake Combined Material Assets

It is recommended to do the following in a setup scene different from your game level scene.

Before organizing `LOD objects` into `clusters`, the developer first needs to define which objects can be baked together by baking the materials on these objects together into atlases and creating `Material Bake Result` assets. As long as all the materials on a mesh have been baked into an atlas, that mesh can be combined with another mesh whose materials have been baked into the same atlas.

Some things to consider when grouping objects are:

- shader
- material
- render type (MeshRenderer or SkinnedMeshRenderer)
- proximity to other objects
- texture tiling
- lighmapping
- static or dynamic

Consider this example:

- 3 house models (Diffuse, static)
- 3 tables (Diffuse, static)

- 5 chairs (Diffuse, static)
- 3 doors (Diffuse, dynamic)
- 6 windows (Transparent, static)
- 12 glass goblets (Transparent, static)
- 4 NPC skinned meshes (Diffuse, dynamic)
- 6 tools/weapons to be used by the NPCs (Diffuse, dynamic)

A reasonable way to bake these materials would be:

- Group 1 (Diffuse shader, static, MeshRenderer)
  - house models
  - tables
  - chairs
- Group 2 (Transparent shader, static MeshRenderer)
  - windows
  - glass goblets
- Group 3 (Diffuse shader, dynamic, MeshRenderer or SkinnedMeshRenderer)
  - doors
- Group 4 (Diffuse, dynamic, SkinnedMeshRenderer)
  - NPC skinned meshes
  - tools/weapons

IMPORTANT: All Renderers that are part of an `LOD component` must have all of their materials baked together.

Once these groupings have been determined, the developer creates a "`Mesh and Material Baker`" for each group, and bakes the materials in each group together into a `Material Bake Result` asset and set of atlases.

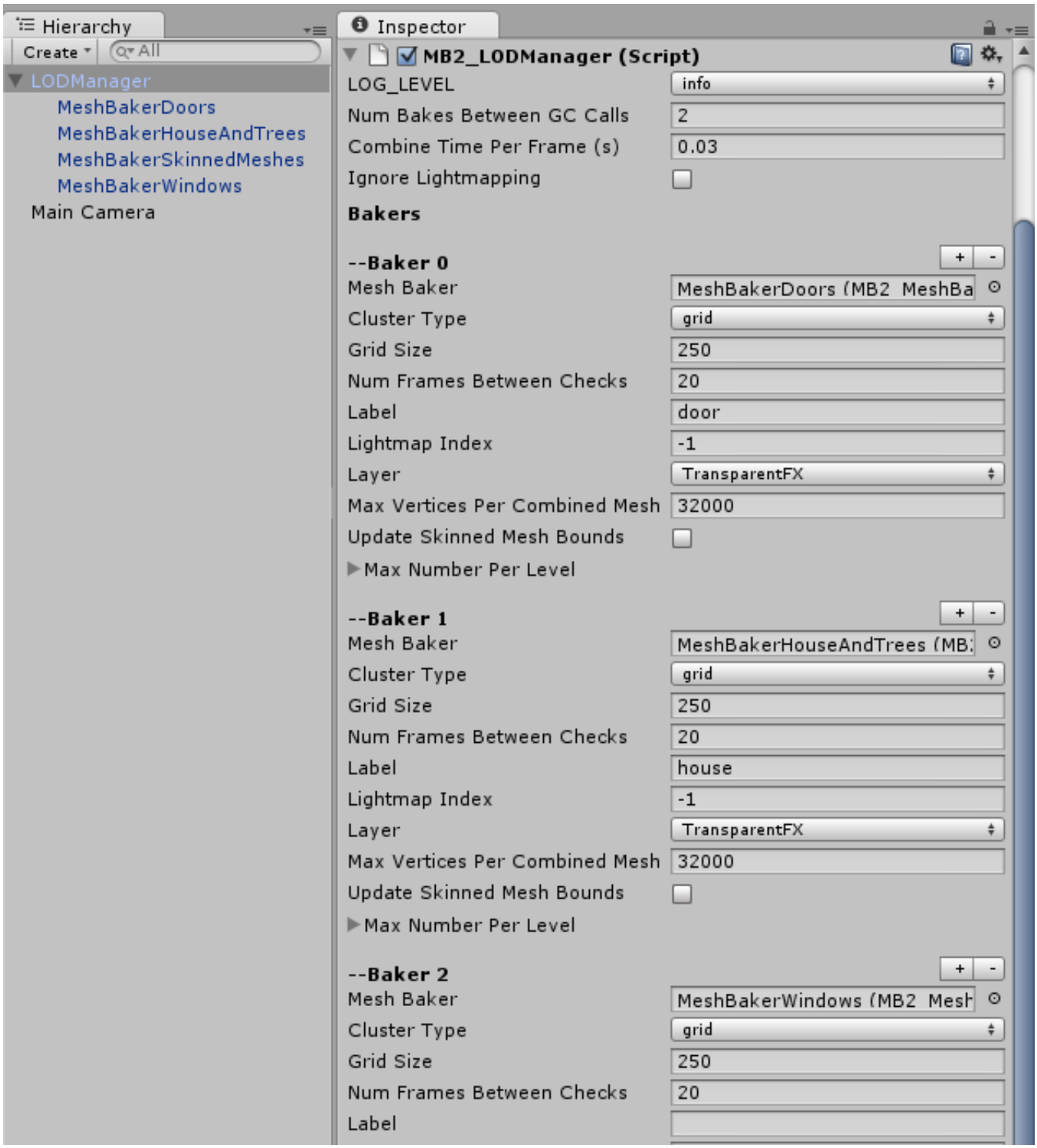
The objects added to the `Material Baker` during this step can be prefab assets or instances. The output of this step is the `Material Bake Result` asset and atlases.

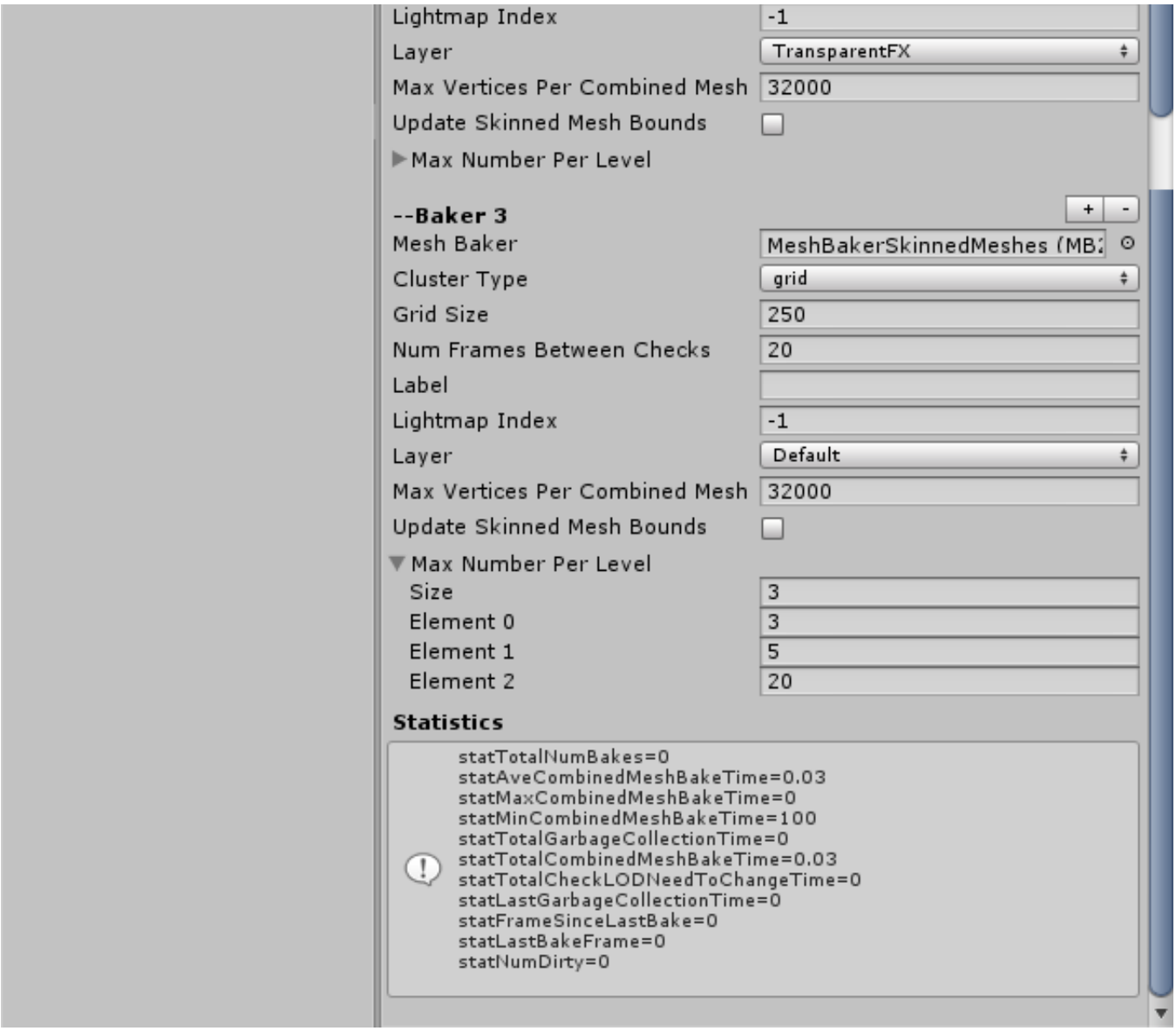
Although meshes are not baked together at this time, it is a good idea to do a test bake, to check that the meshes will combine correctly.

Step 2

The developer next adds a single (there must be only one) `LODManager` to the setup scene. Drag the `LODManager` prefab into the scene or create a game object and add a `MB2_LODManager` component.

Now add `bakers` in the `LOD Manager` inspector by clicking the + sign or the "`Add New Baker`" button. There must be one `baker` for each `mesh and material baker` created in step 1. Each `baker` consists of a reference to a `mesh baker` object and some extra information (clustering scheme, size of cluster, max verts per mesh, etc..)





Drag the `Mesh and Material Baker` game objects to the `LOD Manager` so that they are all children of the `LOD Manager`.

These `besh baker` components are not used directly. They are prototypes who's settings will be cloned by `MB2_MultiMeshCombiner` instances in each cluster.

Drag each `mesh baker` to one of the `bakers`, and configure the `baker`.

Cluster Type: If the meshes are skinned meshes or can move, then the clustering scheme must be `simple or moving`.

Grid Size (grid cluster only): If your game world is endless and you are adding and removing chunks as the player moves around, then it is a good idea to set your `grid size` to be the same as your chunk size. Other things to consider are frustum culling and mesh density. If

your `grid size` is much larger than your camera frustum, then you may be sending a lot more geometry to the graphics card per drawcall than necessary. On the other hand, if there are only a few meshes per cluster, there may be more overhead than necessary.

Num Frames Between Checks: It is expensive to check whether an LOD needs to change every frame. There is a big performance benefit to setting this to a large value (every half second or even once a second). Note that this value is used for clusters close to the camera. Clusters that are more distant are checked at a multiple of this value depending on how far they are from the camera.

Label: The label field is used to resolve conflicts if an `LOD object` can be mapped to more than one `baker`.

Max Vertices Per Combined Mesh: The maximum mesh size in Unity is 64k vertices. Meshes this big take a long time to bake on most platforms, and could cause a stutter in frame rate. Lowering the maximum mesh size results in more meshes/drawcalls. However, add and remove operations on smaller meshes are much faster. For mobile, 12k would be a reasonable starting point.

Create a prefab of your configured LODManager object.

STEP 3

Add an LODCamera script to your player's camera. There can be more than one camera. LODs will adjust their level based on the closest camera.

Step 4

Now you are ready to setup `LOD objects`. The hierarchy of each `LOD object` should look something like this:

- Game Object (LOD component goes here)
  - LOD0 Game Object (has at least one MeshRenderer or SkinnedMeshRenderer somewhere in its children)
  - LOD1 Game Object (has at least one MeshRenderer or SkinnedMeshRenderer somewhere in its children)
  - LOD2 Game Object (has at least one MeshRenderer or SkinnedMeshRenderer somewhere in its children)

Note that all the levels in each LOD component must map to the same `mesh baker` (all materials must have been baked together).

Configure the `LOD component` by dragging the MeshRenderer or SkinnedMeshRenderer to its appropriate slot in the list of `LOD levels`, and by setting the `screen percentage` (ie: .25 means that the largest dimension of the mesh takes up one quarter of the screen). You do not specify directly which `baker` will bake this `LOD object`. The `LOD manager` assigns each `LOD object` to the correct `baker`, based on its material(s), renderer, light map index and label.

If you want to ensure that a particular `LOD object` gets baked by a particular `baker`, then you can add a unique **label** to that baker and specify the `baker's label` in the LOD component `label` field.

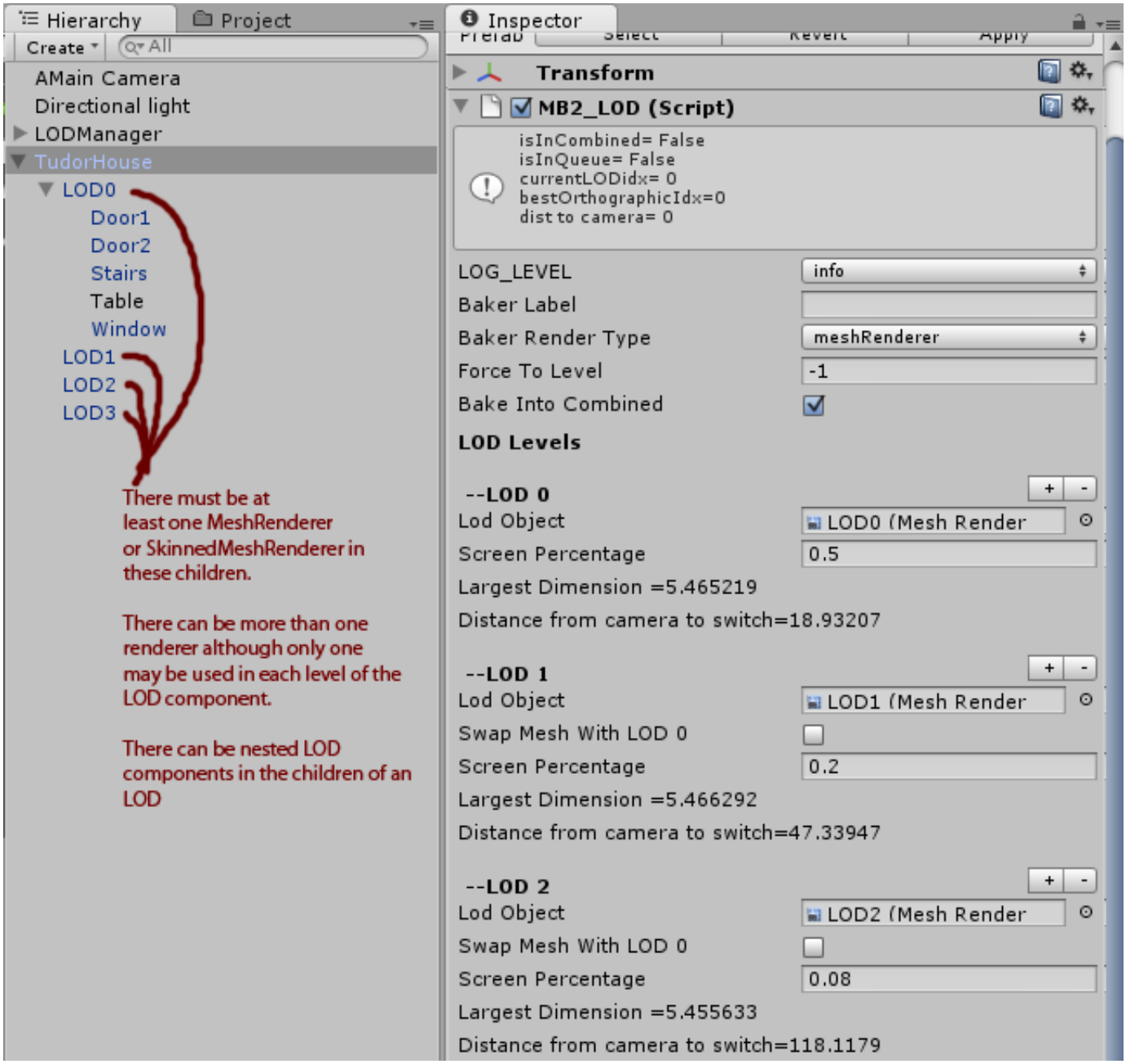
For SkinnedMeshes, set the `render type` on the LOD component to `SkinnedMesh`. Make sure that the `baker` that will bake this mesh also has its render type set to `SkinnedMesh`, and that the `mesh baker` component used by that baker has its `renderer` field set to `SkinnedMeshRenderer`.

Important feature: LOD objects can be nested!

For example you could have a house with several levels of detail. The closest level of detail could contain child game objects for the interior of the house along with furniture and characters. The furniture and characters can in turn have LOD components attached. They will be baked when enabled and removed when disabled.

As you set up each LOD object you can test it by pressing play and moving the camera closer to and further from the LOD object you are testing. Check the console for warnings and errors.

Once each LOD is set up create a prefab for that LOD object.



Your setup is now complete!

```
Add Prefabs To Your Game Level
```

You can now easily add your LOD objects to any of your game levels by:

- Adding your configured LODManager prefab
- Adding MB2_LODCamera scripts to your cameras
- Dragging as many of your prefabbed LOD Objects into your scene as youe wish.

LOD CONFIGURATION OPTIONS

Baker Label: Used to resolve conflicts if an LOD matches more than one baker. Match this label to that of a baker to force this object to be baked by a particular baker.

Baker Render Type: This LOD must match a baker with this render type.

Force To Level: Forces the LOD into this level.

Bake Into Combined: If this is not checked then LOD will take place but the visible mesh will NOT be baked into a combined mesh.

Swap Meshes With LOD 0: If checked then when this level becomes active, it's mesh will be copied into LOD 0 which will be made active. This is particularly useful for skinned mesh renderers so that animations and scripts on LOD 0 will be continuous. You can remove animations and other scripts on this level since this level will never be active. All meshes need to use the same bone hierarchy for this option to work.

Skinned Mesh Renderer Mob Considerations

If you wish to combine many skinned meshes into a mob there are a few considerations:

- If the skinned meshes are in different clumps you may want to use several bakers to keep the combined meshes separate and use labels to map the meshes to the correct baker. This will allow culling to occur.
- You will need to update the local bounds of the combined skinned mesh renderer. There is an option for updating based on bounds, but this is expensive since it is calculated every frame. It may be more efficient to set the local bounds to a large value once and leave it.
- It is hard and expensive to keep animations synchronized as LOD switches occur. An excellent option is to use the "Swap Meshes With LOD 0" option.

IMPORTANT NOTE ABOUT DESTROYING OBJECTS

It is only safe to call Unity's Destroy method on objects with `MB2_LOD` components attached if they are not in the combined mesh (isInCombined flag is false). This is because the LOD may have meshes in the combined mesh which can't be removed if these meshes are destroyed. The easiest way to Destroy an `LOD object` is to call:

```
MB2_LODManager.Manager().LODDestroy(myLOD.GetComponent<MB2_LOD>());
```

This method defers actual destruction until the `LOD objects` meshes are removed from the combined mesh.

WHAT HAPPENS AT RUNTIME

Each scene will have one (and only one) `LOD_Manager` component. The `LOD objects` register with the `LOD Manager` at awake and is mapped to a baker based on its materials, render type, lightmap index etc… On Update, the `LOD_Manager` checks LODs to see if the mesh that is being displayed needs to change. This checking is distributed evenly across frames. If an LOD does need to change, the LOD component registers a change action with the cluster that contains it.

The `LOD manager` organizes the scene into clusters for each baker. Currently there are two clustering schemes:

- **Simple**: All objects that are assigned to this baker are baked into one huge global combined mesh. This is the only clustering scheme allowed for moving objects and skinned meshes.
- **Grid**: Space is divded into cubic cells and all objects assigned to this baker within each cell are baked together. Objects to be combined must not move.

In each LateUpdate, the `LOD manager` prioritizes the queue of dirty `clusters`. It adds, removes and updates meshes as needed from the combined meshes. The baking can be distributed across several frames if baking takes a long time.

TROUBLESHOOTING AND PERFORMANCE MONITORING

Try setting the `LOG LEVEL` field of the `LOD Manager` to "`debug`" or "`trace`".

Try setting the `LOG LEVEL` field on individual `LOD objects` to "`debug`" or "`trace`".

The `LOD Manager` inspector and `LOD object` inspectors displays useful stats and status information such as:

- How frequently baking is happening
- How much time baking takes
- How long garbage collection takes, etc.

TIP: To display these stats on a device, attach a `MB2_LODManagerGUI` component to the LOD manager.

If the LOD_Manager debug level is set to “debug” then a frame by frame breakdown of the time taken will be printed to the console.

These statistics should provide guidance when adjusting the performance settings.

PERFORMANCE CONSIDERATIONS

The following things need to be considered for performance tuning:

1. How much time is spent checking if LODs need to change
2. How long does an individual bakes take
3. What is the total baking time
4. How long is garbage collection taking

PERFORMANCE TUNING

Num Frames Between LOD Checks: This is probably the most powerful performance tuning setting. It is usually not necessary to check if LODs need to be changed on each object, every frame. It takes just as long to update one mesh in a combined mesh as it does to update all the meshes. It is much better to wait and update several LODs at once in a batch than to update the combined mesh with every LOD change.This value can likely be 1/4 sec, 1/2 sec, 1 sec or even longer. Clusters close to the camera use this value. Clusters that are more distant will use a multiple of this value depending on how far they are from the camera.

Bake Into Combined: Consider disabling this on the Level 0 meshes of your LODs. Often the LOD 0 mesh has many more vertices than the higher level meshes and only becomes visible when the character is very close. There is little performance benefit to baking these meshes into combined meshes because they use a lot of space in the combined meshes, take a long time to bake and only a few are visible at a time. Most of the performance benefit of Mesh Baker LOD comes from baking hundreds of tiny distant meshes. There is little gain in frequently baking a few large meshes close to the camera.

Num Bakes Between GC Calls: Combining meshes requires a lot of array allocations. It is a good idea to run the garbage collector regularly to avoid a stutter. Check the stats window on the `LOD Manager` to see how long garbage collection is taking, and set this to the largest value you can

reasonably get away with.

Combine Time Per Frame Threshold: Mesh Baker LOD will try to distribute bakes across several frames to avoid stuttering. The `LOD Manager` will continue baking until this threshold is exceeded. If there is a lot of baking to do, then the load will be distributed across several frames. A reasonable setting would be slightly less than 1 / desiredFrameRate.

Max Vertices Per Combined Mesh: The maximum mesh size in Unity is 64k vertices. Meshes this big take a long time to bake on most platforms and will cause a stutter in framerate. Lowering this means more meshes/drawcalls but reduces the atomic bake time and baking can be distributed across several frames. For mobile, keeping this bellow 10k or even 5k is reasonable.

Grid Size: The size of the cubic cells for the grid based clustering. Sometimes all the meshes in a cluster need to be baked so it is ideal for a cluster to be small enough that it contains only a few combined meshes. On the other hand if there are two few LODs in a cluster then the combined meshes will be small and there will be many drawcalls. Clusters should also be smaller than the view fustrum so that excess geometry that would normally be culled is not included in drawcalls. A final consideration is that distant clusters are checked less frequently for LOD changes. There is a benefit to having several rings of clusters in a scene with many objects.

RUNTIME USE

The following calls can be made to the LODManager at Runtime.

```
public void AddBaker(BakerPrototype bp)
```

Adds a baker to the LODManager.

```
public void RemoveBaker(BakerPrototype bp)
```

Adds a baker to the LODManager.

```
public void TranslateWorld(Vector3 translation)
```

If your world is very large and you wish to translate every model and camera in the world to avoid floating point precision problems you can use the LODManager call TranslateWorld. This updates the the cluster bounds, resets the lod positions if the world has been translated and moves the combined meshes. This does NOT move the LOD game objects. These should be moved before TranslateWorld is called. This should be called in LateUpdate, after all LODs have been checked. This is expensive so don't call it frequently.

```
public void AddCamera(MB2_LODCamera cam)
```

Adds a camera to the LODManager

```
public void RemoveCamera(MB2_LODCamera cam)
```

Removes a camera from the LODManager

```
public void ForceBakeAllDirty()
```

Bakes all dirty clusters. It is a good method to call to clear the queue when there is a pause in gameplay.

EDIT