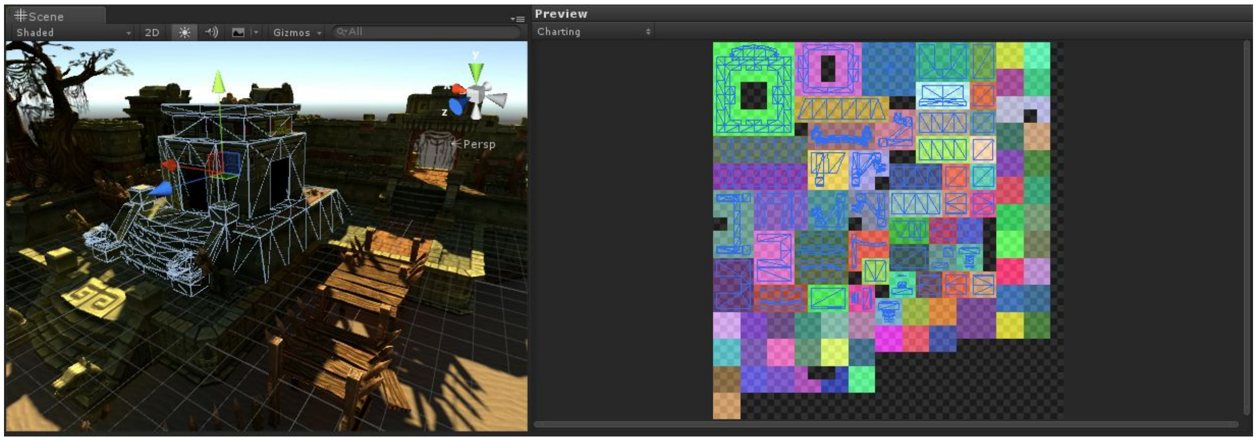
Global Illumination UVs in Unity 5

There are two sets of UVs that are particularly important for global illumination in Unity 5; the UVs used for precomputed realtime GI and the UVs used for baked lightmaps. This document describes the flow of those UV sets.

How can I see my UVs

It is generally important to be able to view the UVs that are being used, so in Unity 5 we have introduced some visualisation for this. For convenience first enable "`Lighting [panel] -> Auto`". This will make sure your bake and precompute are up-to-date and also output the data that is needed to view the UVs. Wait for the process to finish.
To see the precomputed realtime UVs:

- Select the instance
- Open "`Lighting [Panel] -> Object`"
- In the "`Preview`" area, select "`Charting`" from the dropdown.

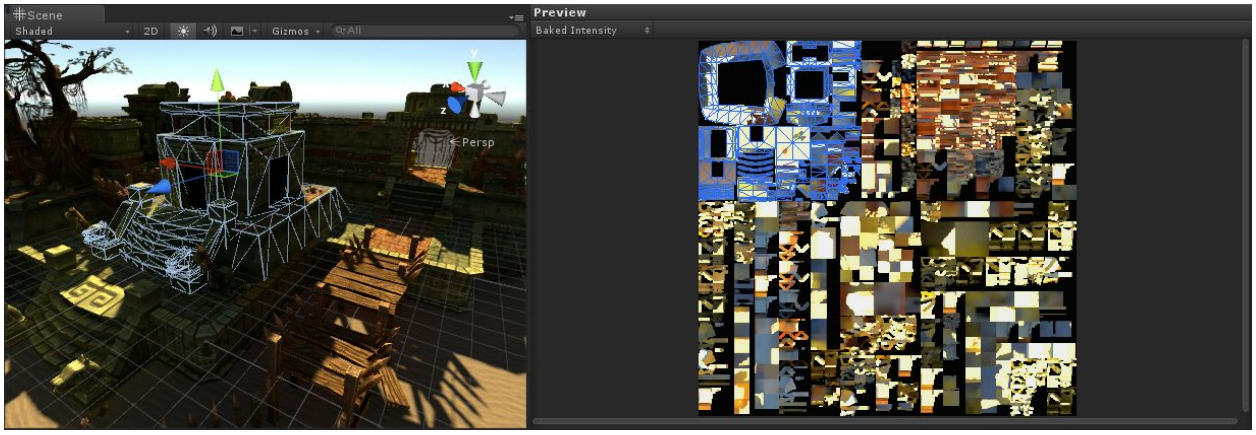


This will show the UV layout for the realtime lightmap belonging to the selected instance of this mesh. The differently colored areas are the charts and the UVs of the selected instance are overlaid. Grey stuff is unoccupied areas of the lightmap. Generally more than one instance lives in a realtime lightmap so some of the charts will not belong to the model.

NOTE: There is no direct correspondence between realtime and baked lightmaps. Two instances in the same realtime lightmap may be in two different baked lightmaps and vica versa.

To see the baked UVs:

- Select the instance
- Open "`Lighting [Panel] -> Object`"
- In the "`Preview`" area, select "`Baked Intensity`" from the dropdown.



As you can see the UVs are quite different. This is because the requirements for baked and precomputed realtime UVs are different.

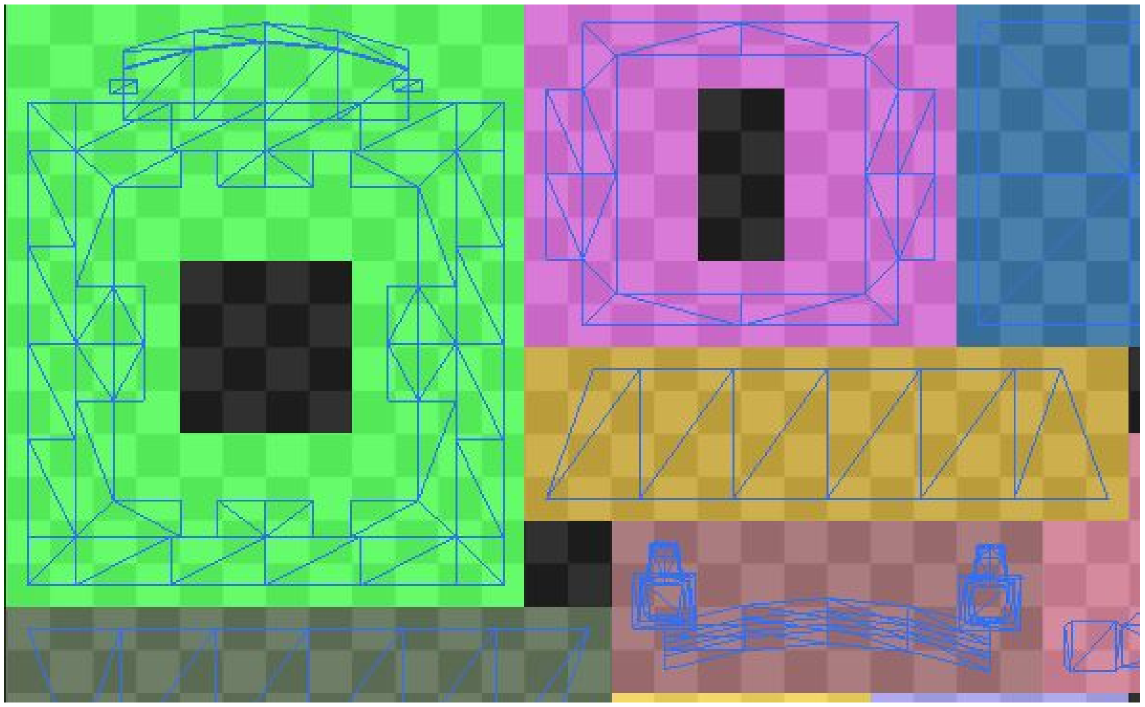
Precomputed realtime UVs

Let me say first, you will **never** get the same UVs for precomputed realtime GI as for baked GI (even if you tick "`Preserve UVs`").

*You will **never** get the same UVs for precomputed realtime GI as for baked GI.*

If you did you would be seeing horrible aliasing such as light or dark edges in unexpected places. Why is this? It is simply because the resolution of realtime lightmaps is intentionally low such that it is feasible to update them in realtime. This is fine since it only stores indirect lighting which is generally *low frequency*. The high frequency content i.e. the direct light and shadows is rendered separately using standard realtime lighting and shadow-maps.

With low resolution lightmaps you would be constantly battling bleeding issues where charts would be sharing texels and thus getting bad lighting. This is solved by repacking the UV charts ensuring a half pixel boundary around them. This way even with bilinear interpolation you will never be sampling *across* charts. The other benefit of charts with a guaranteed half pixel boundary is that charts can be placed right next to each other without problems, so you will be saving wasted lightmap space.



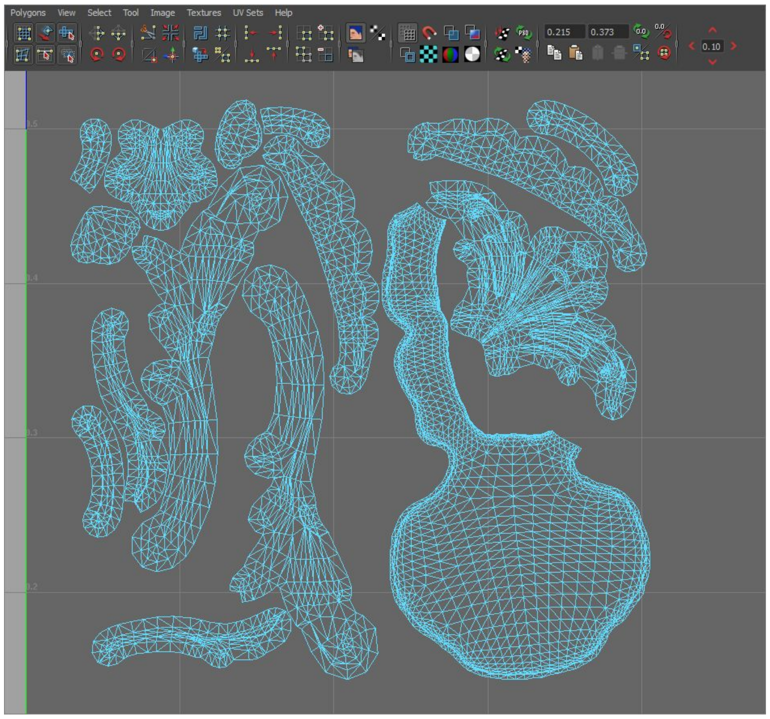
So in summary:

UVs used for precomputed realtime GI lightmaps are ***always*** *repacked.*

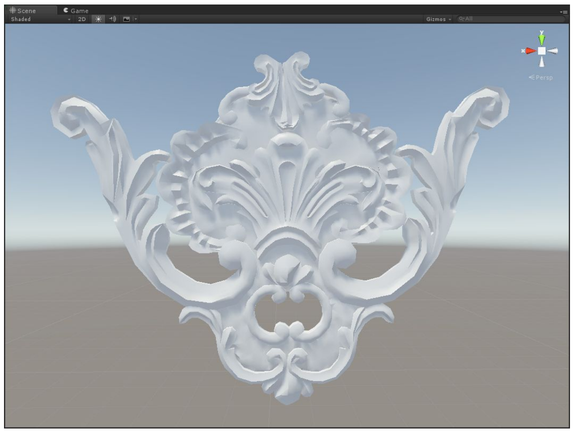
The observant reader will also observe that since repacking guarantees a half pixel boundary around the charts then the UVs will be dependent on the scale and lightmap resolution of the instance. We **cannot** just scale up the UVs to get a higher resolution lightmap and still guarantee this half pixel boundary. The UVs are packed individually with the scale and resolution of the instance taken into account. The realtime UVs are therefore *per instance*.
NOTE: Of course if you have 1000 objects with the same scale and resolution they will share the UVs.

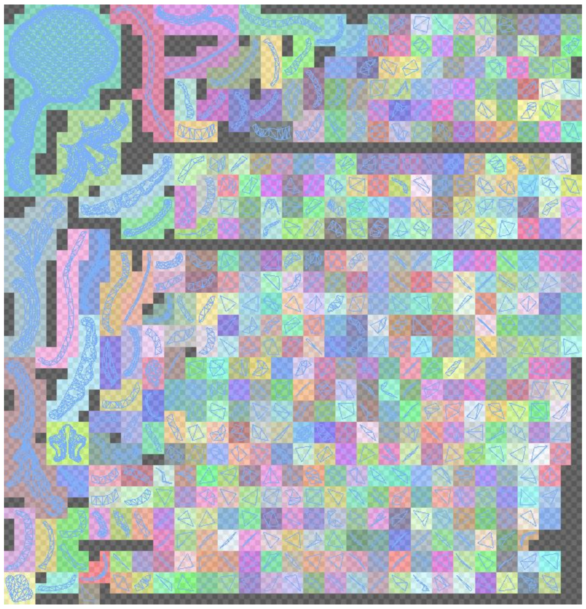
I ticked “Preserve UVs” my UVs still don’t look like they do in Maya!

Because realtime UVs are repacked by Enlighten it is very important how UV charts are detected. By default a chart is defined by a set of connected vertices. However, the Unity mesh importer may introduce extra vertices in places where the mesh has hard edges. These duplicated vertices will create extra islands in your UVs. However, when you bake lightmaps these cuts will normally go unnoticed since the UVs are used directly and not repacked.
The following model is an example of this. The UVs in Maya looks like this:



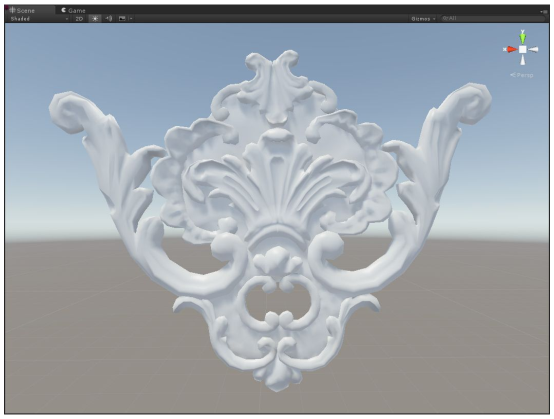
The result when packing with a 40 degree smoothing angle shows a lot of extra charts created by preserving the hard angles in the model:

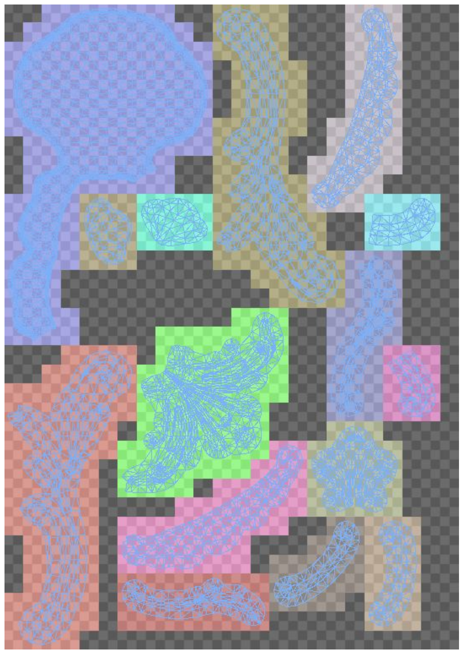




Now if the “`Smoothing Angle`” is set to 180 degrees there will be no cuts made and the UVs will be the same as was authored in Maya. The only difference will be the packing of the charts.

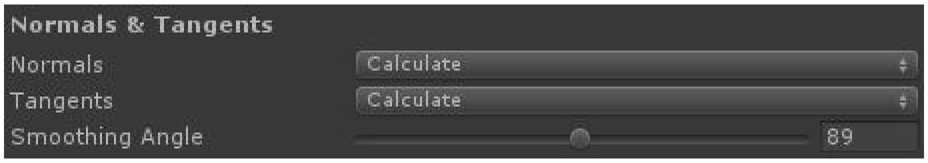
This will differ from Maya:





A high smoothing angle will not preserve hard cuts in the model and both the shading and GI will look different.

The mesh importer settings that relates to this looks like this:



Normals & Tangents

Normals: Calculate

Tangents: Calculate

Smoothing Angle: 89

When normals are set to “`Calculate`”, breaks are made where the angle between adjacent triangles exceed the “`Smoothing Angle`”.

In order to avoid this you can choose to author and import normals. In order to get good results with imported normals, the cuts along hard edges must be made manually. If this is not done both GI and regular shading may look bad.

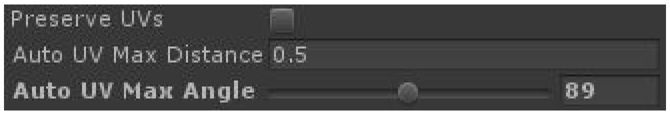
Auto UVs

Now, "`Preserve UVs`" is badly named. It should be called "`Optimize realtime UVs`" and be inverted. What “`Preserve UVs`” does is **skip** the Enlighten auto UV feature. It will **not** let the authored UVs flow straight to Enlighten for the reasons discussed above; repacking is still applied.

The auto UV feature is intended to optimize charting for realtime GI only. It does not affect the UVs used for baking. It intends to reduce the chart count (and thus texel count) in the UV unwrap by simplifying it. This will make lighting more consistent across the model, make the texel distribution more even, and avoid wasting texels on small details. The time taken to do the precompute phase is proportional to the number of texels you feed it. For instance a detailed tiled floor with separate charts for each tile would take up unnecessarily many texels, joining them into a single chart would result in much fewer texels. This would work fine since the lighting stored in the realtime lightmaps is indirect lighting only (there are no sharp direct shadows).

This process **cannot** alter the number of vertices in the model so it **cannot** introduce breaks in the UVs where there isn't a break in the incoming UVs already. So the chart layout will be the same but some of the charts may overlap or be merged in areas where it is deemed to not affect the indirect lighting badly. **NOTE: However, currently there is a bug in Unity that causes chart splits to happen even when "`Preserve UVs`" is enabled. This is because the mesh importer introduces duplicated vertices on hard edges, and this confuses the chart detection that will see these as separate UV islands. A fix for this is coming soon.**

The controls for the Auto UV feature can be found in "`Lighting [Panel] -> Object`". You have to untick "`Preserve UVs`" to enable them:



Preserve UVs

Auto UV Max Distance 0.5

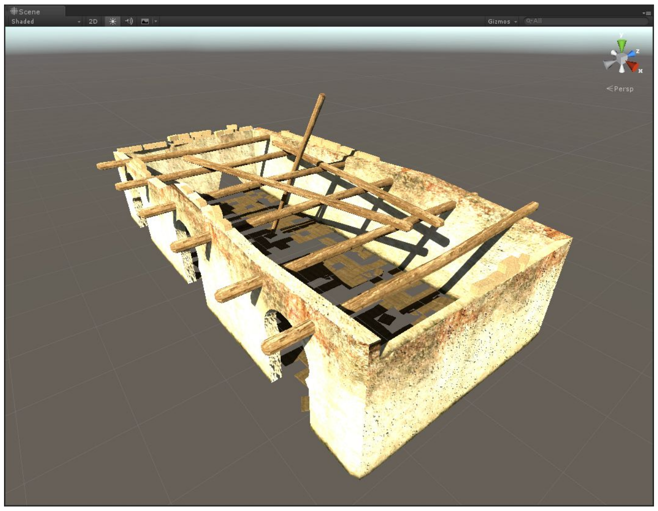
Auto UV Max Angle 89

The settings controls when the charts can be merged (Auto UV docs).

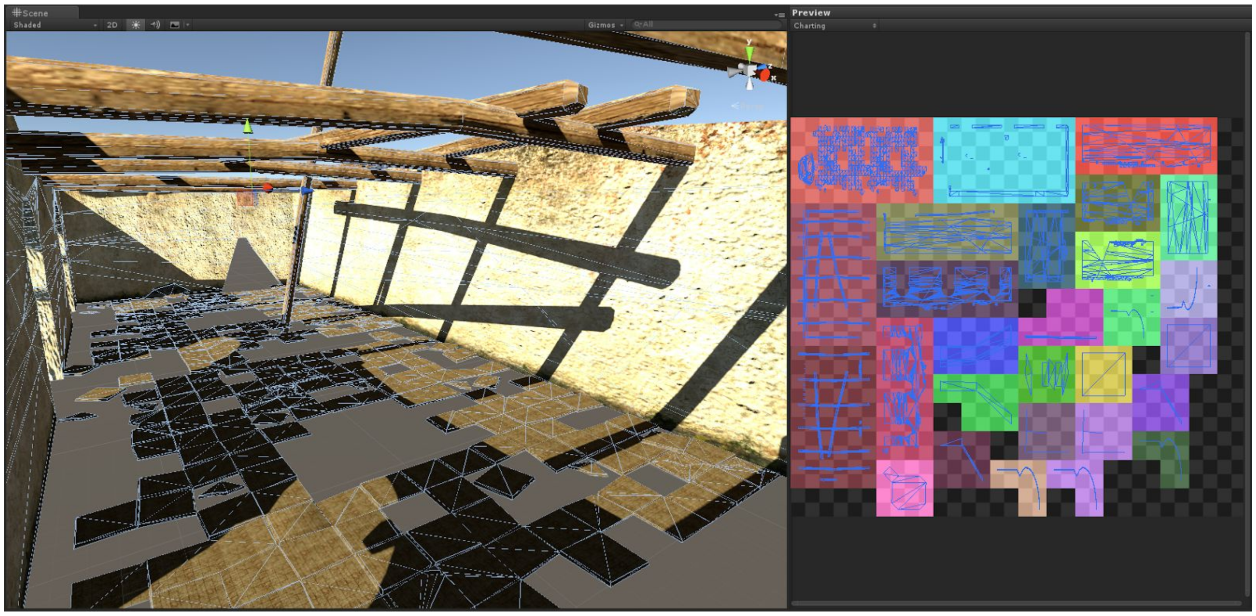
- "`Auto UV Max Distance`": Charts will only be simplified if the worldspace distance between the charts is smaller than this value.
- "`Auto UV Max Angle`": Charts will only be merged if the angle between the charts is smaller than this value.

The settings are intended to avoid simplifying charts when they are pointing in generally different directions or are far apart.

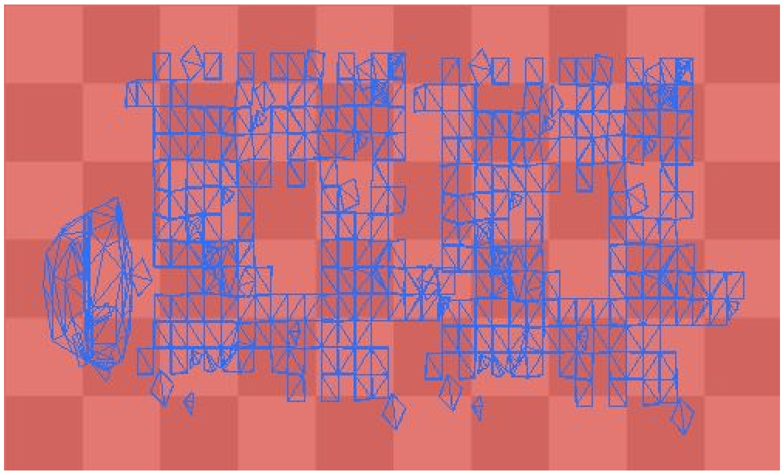
To show an example I have taken a nice model from the Desert Ruins asset from the asset store:



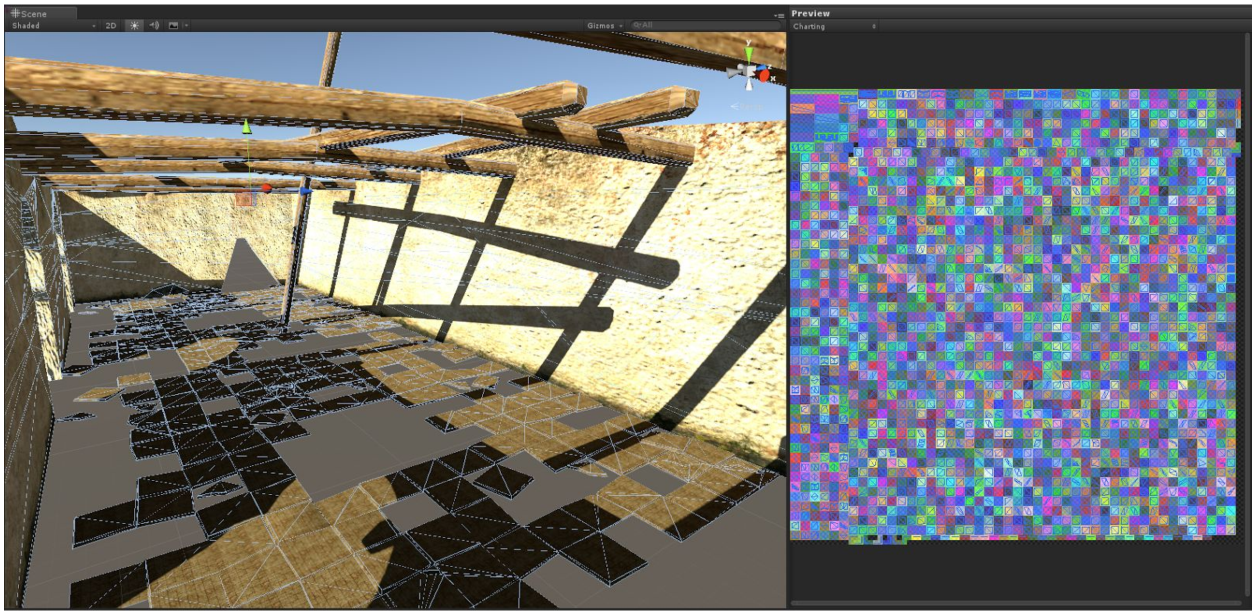
It is using default parameters and a single realtime texel per unit. The model is roughly 9 units long. The realtime UVs generated for this model looks like this when using the Auto UV feature:



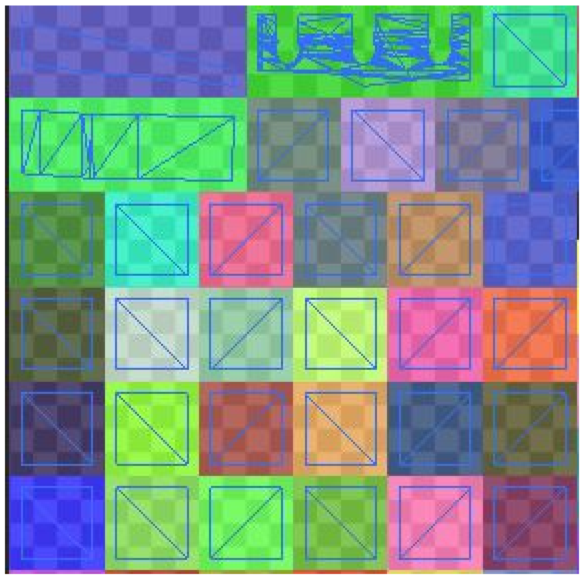
Notice especially the little tiles on the floor have been nicely packed to a single chart with a sensible resolution considering the chosen texel density and instance size:



When packing without the Auto UV feature the UVs come out like this:



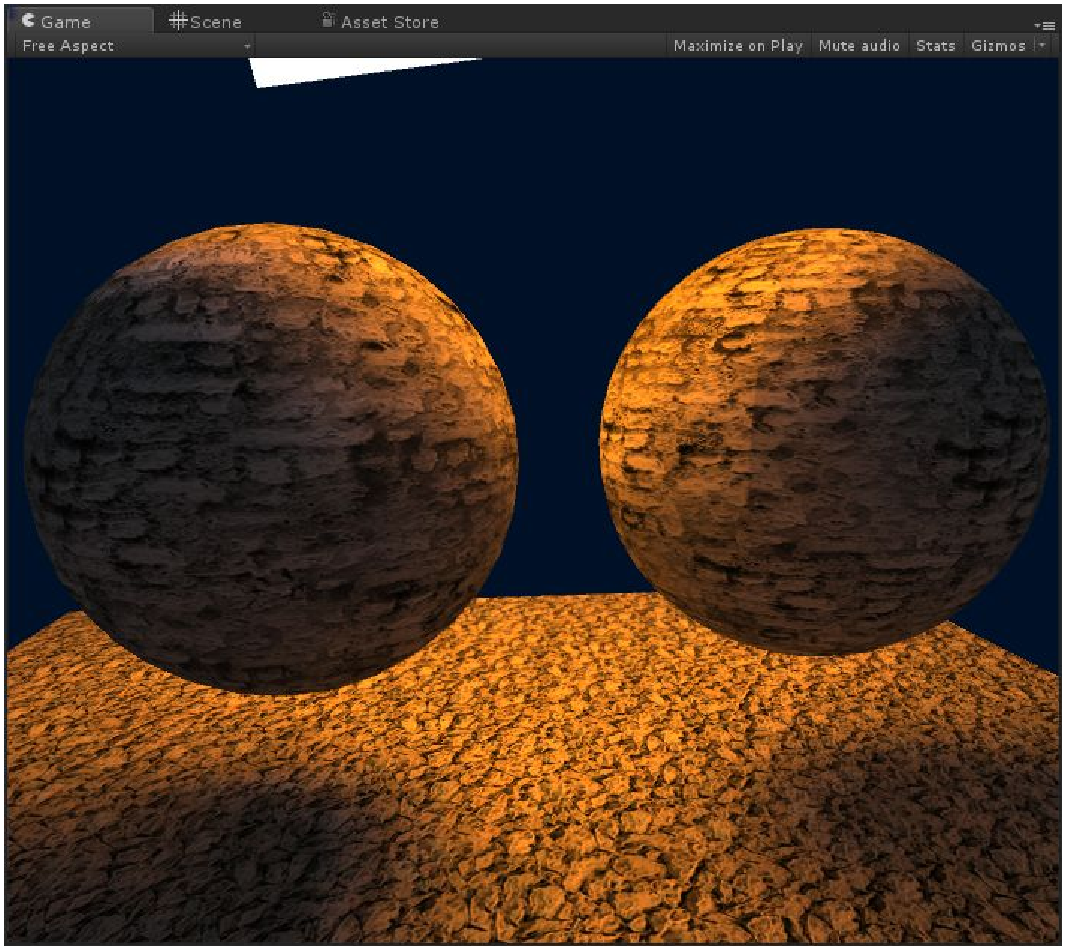
This generates an explosion of little charts because the charts are split excessively in the authored UVs supplied by the model. Without Auto UVs none of these charts can be merged and each UV island is awarded a 4x4 pixel block of its own regardless of its size. Here is a subsection of the UVs:



The wall sides still get a sensible resolution of 10x4 texels, but the little tiles gets a disproportionate 4x4 texels each. The reason that the minimum chart size is 4x4 is that we want to be able to stitch against the chart on all 4 sides and still get a lighting gradient across the chart. Regardless Enlighten requires a minimum of 2x2 texels per chart so the problem would still remain even if we allowed charts smaller than 4x4 texels.

Getting chart edges to stitch for realtime GI - aka avoid bad seams between charts

The realtime lightmaps support chart stitching. What this does is that it ensures that the lighting on adjacent texels in different charts is consistent. This is useful to avoid visible seams along the chart boundaries. With large texel sizes the lighting on either side of a seam may be quite different and this is not smoothed out by filtering since the texels are not adjacent. A seam is visible on the right sphere even when heavily textured since it has not been stitched:

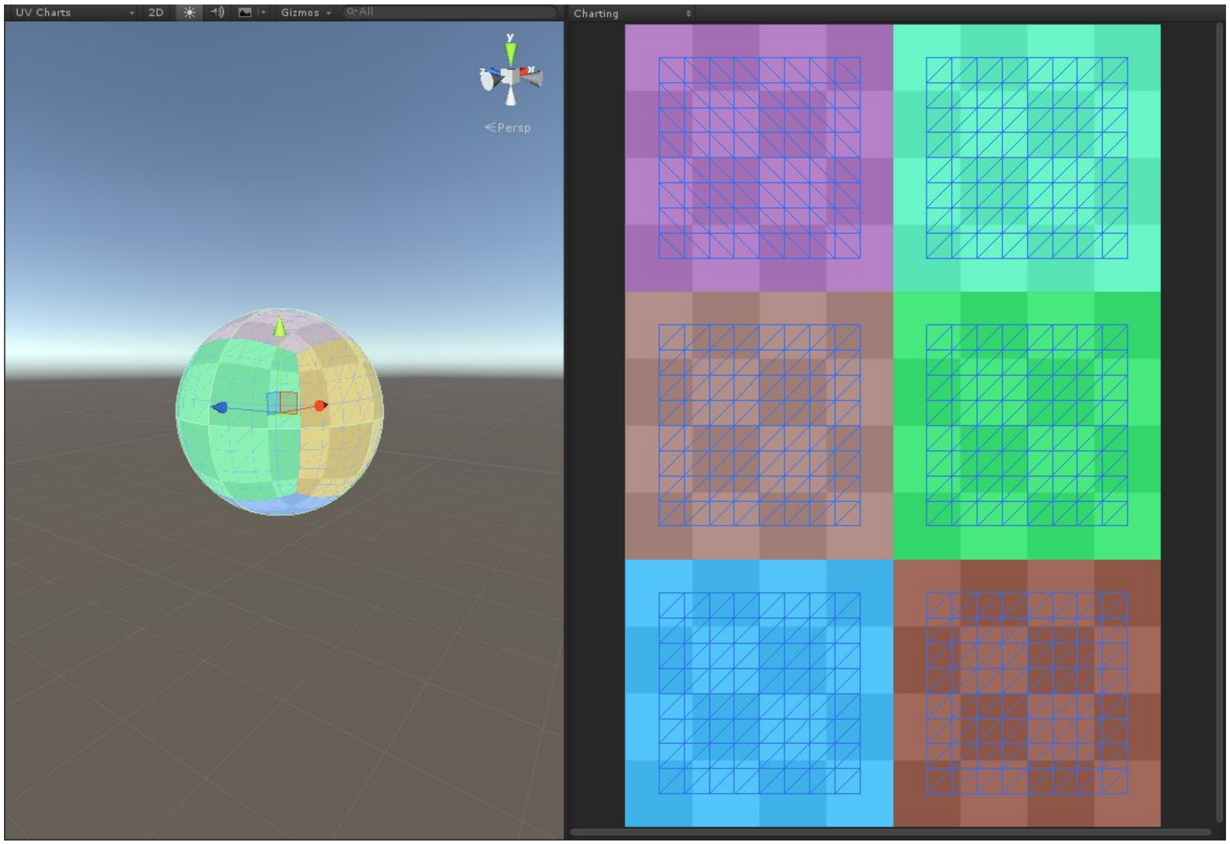


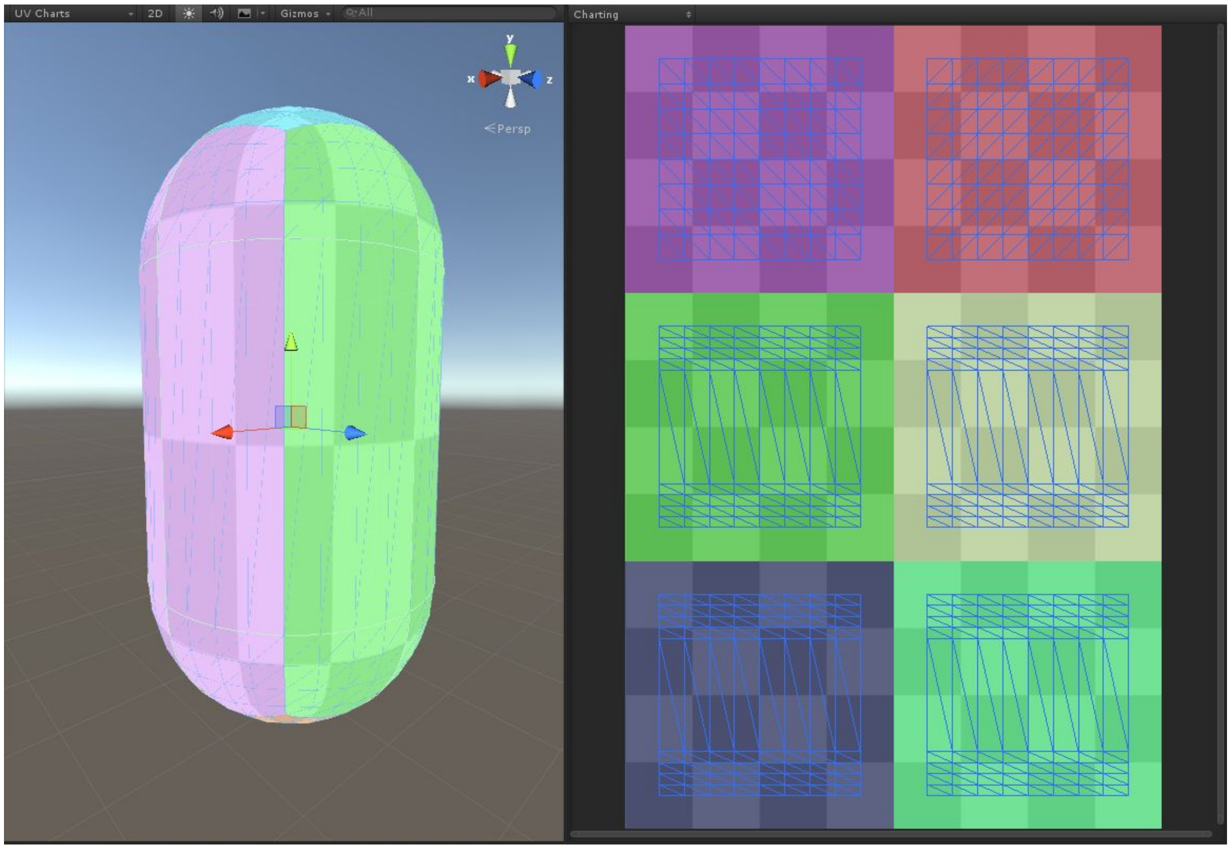
Stitching is on by default. If you think it is causing some unwanted issues and want to disable it, then you have to apply lightmap parameters to the instance in question (LightmapParameters docs) and untick "Edge Stitching".

In order to get charts to stitch nicely there are a few criteria that candidate edges must adhere to:

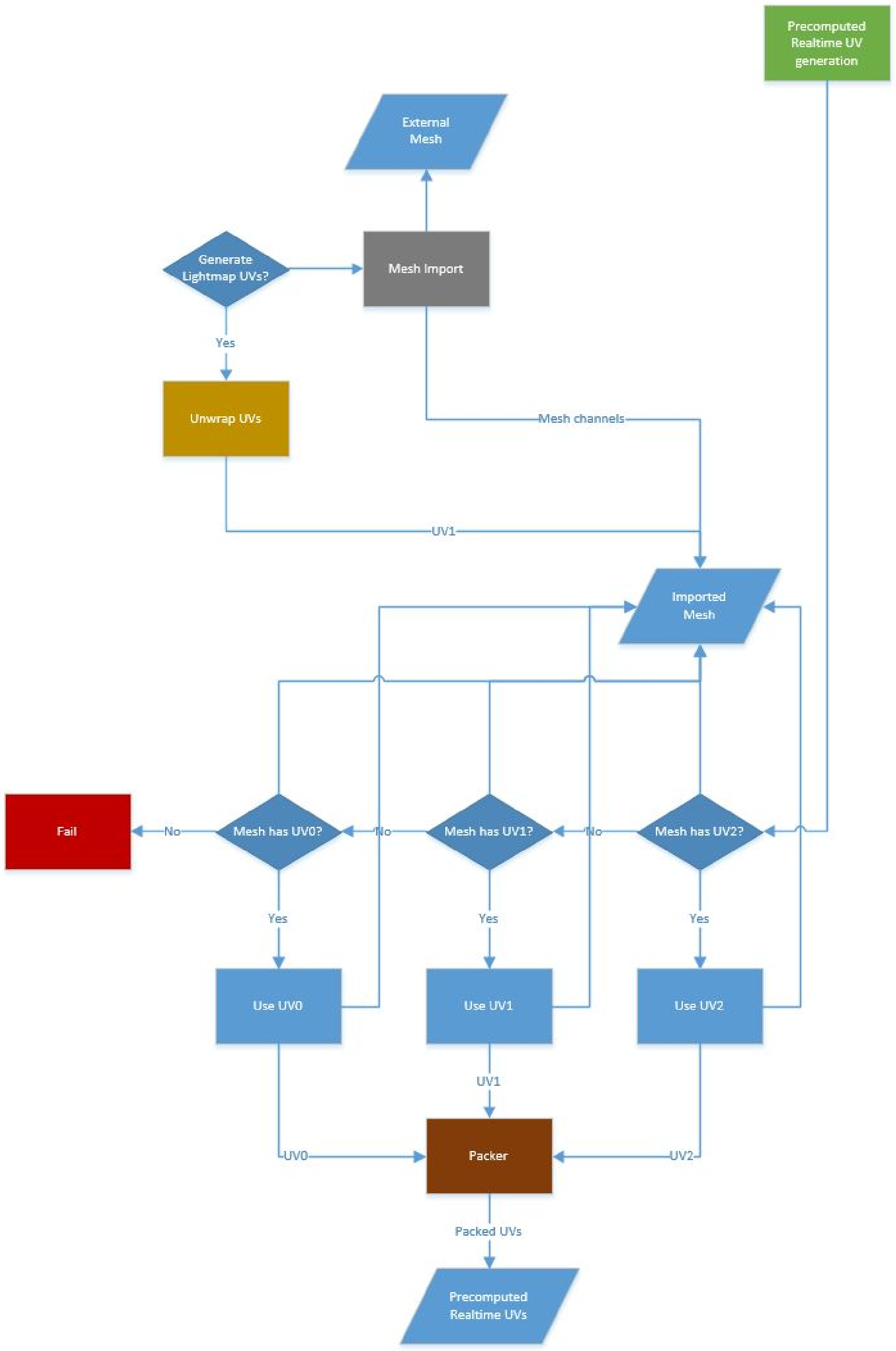
- "`Preserve UVs`" must be enabled so the charts are not simplified by the Auto UV feature.
- The charts must be in the same mesh.
- The edges must share vertices.
- The edges have to be horizontal or vertical in UV space.
- The edges must have the same number of texels (usually follows from the two preceding criteria).

This is incidentally how the builtin sphere, capsule and cylinder are authored. Notice how the charts line up:


UV Charts
2D
Gizmos
All
Persp
Charting


UV Charts
2D
Gizmos
All
Persp
Charting

Precomputed Realtime UVs flow

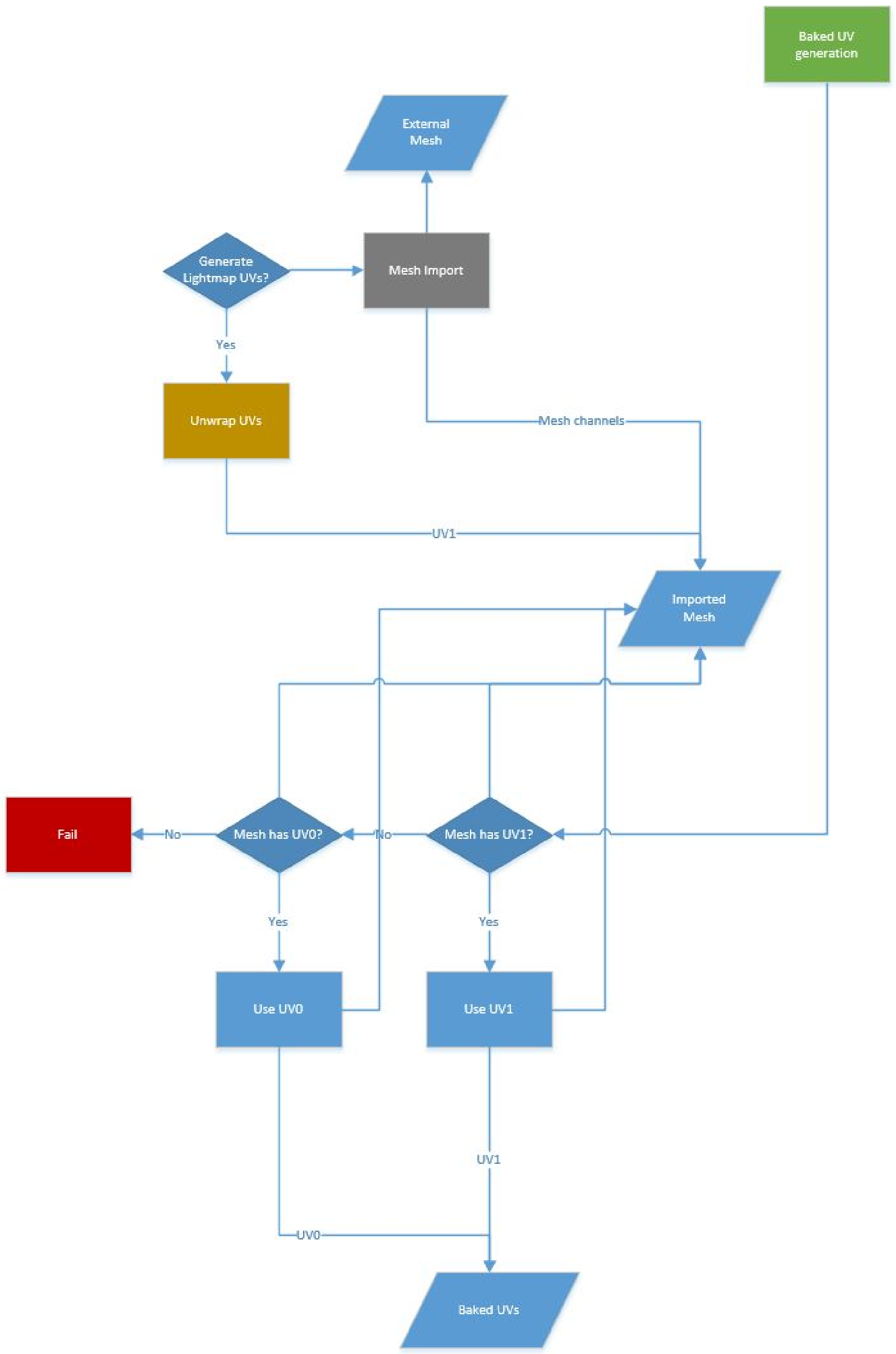


So the main takeaway you get from the UV flow for realtime UVs is that you get to author your own specialized UVs for realtime if you put them into UV2. Falling back to UV1 (baked lightmap UVs) or UV0 (texturing UVs) respectively if needed. The baking will not attempt to use those, it will default to UV1 and fall back to UV0. NOTE: The packer will still need to pack them in order to ensure that they have the half pixel boundary.

Baked UVs

Baked UVs are extracted from the imported mesh and used “as-is”. They are scaled to accommodate the desired resolution. The baking pipeline will use UV1 as the lightmap UVs, if these do not exist it will fall back to UV0. The mesh importer in Unity can generate a lightmap unwrap for you if you tick “`Generate Lightmap UVs`” (docs).

Baked UVs flow


Baked UV generation
External Mesh
Generate Lightmap UVs?
Mesh Import
Yes
Unwrap UVs
Mesh channels
UV1
Imported Mesh
Fail
No
Mesh has UV0?
No
Mesh has UV1?
Yes
Yes
Use UV0
Use UV1
UV1
UV0
Baked UVs