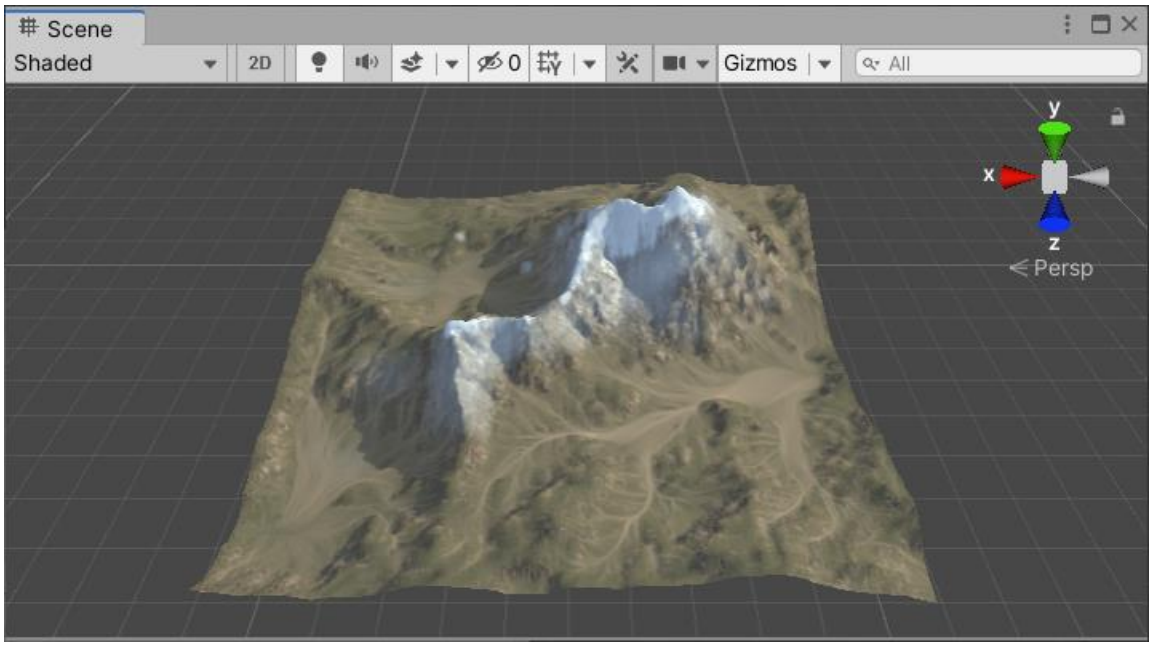
Terrain To Mesh

amazingassets.world

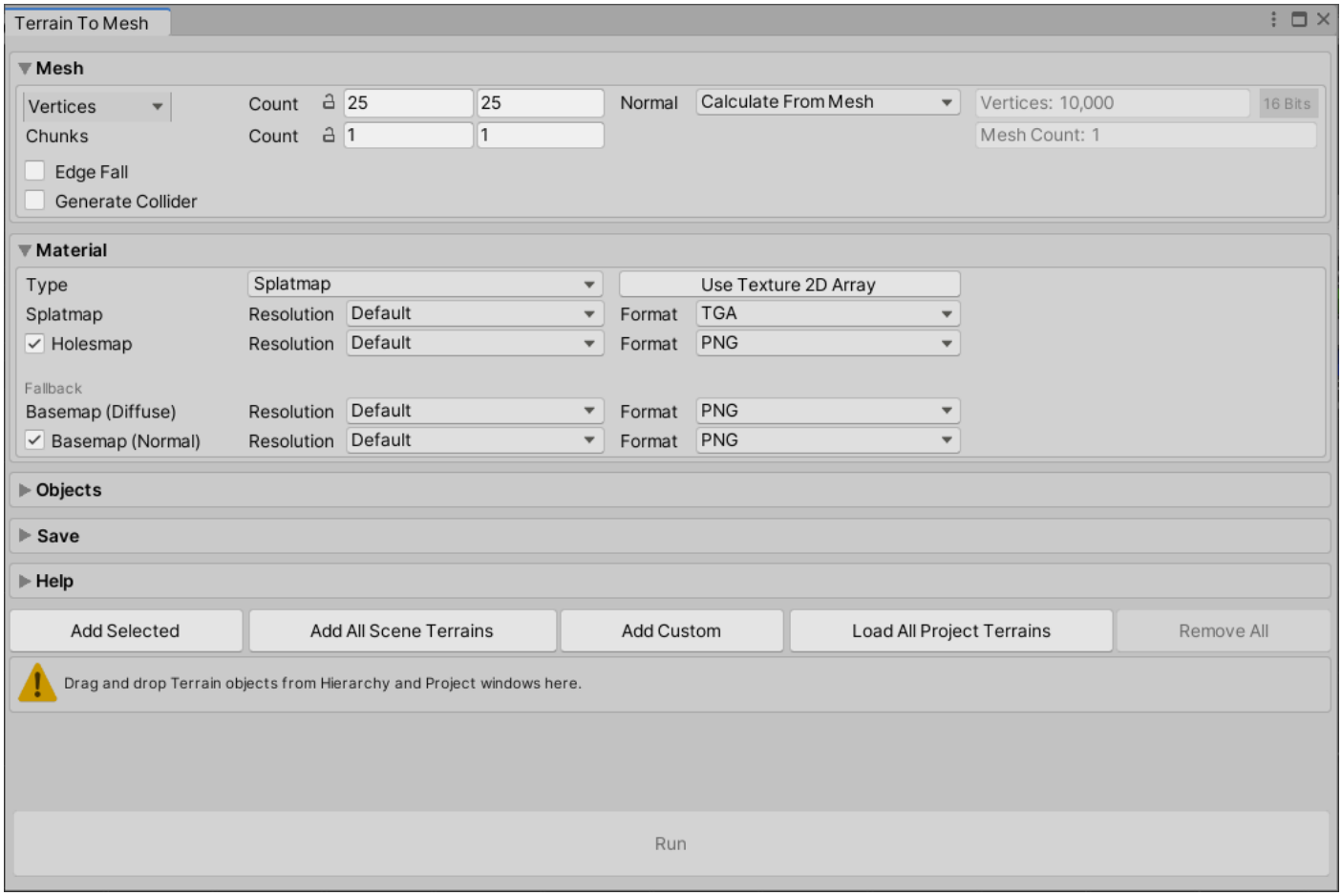
TABLE OF CONTENTS

QUICK START

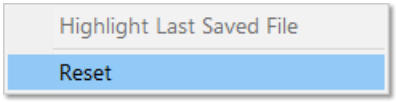
Open **Quick Start** scene from **Terrain To Mesh \ Example Scenes** folder. Scene contains simple Unity terrain only.



Open Terrain To Mesh (TTM) editor window using **Unity Main Menu \ Windows \ Amazing Assets \ Terrain To Mesh**.

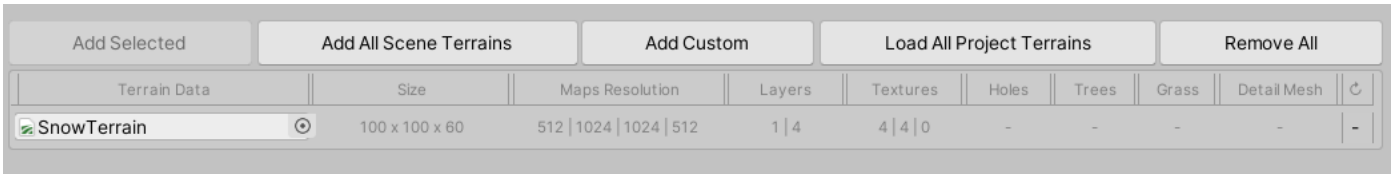


Use context menu to reset window and load default settings.

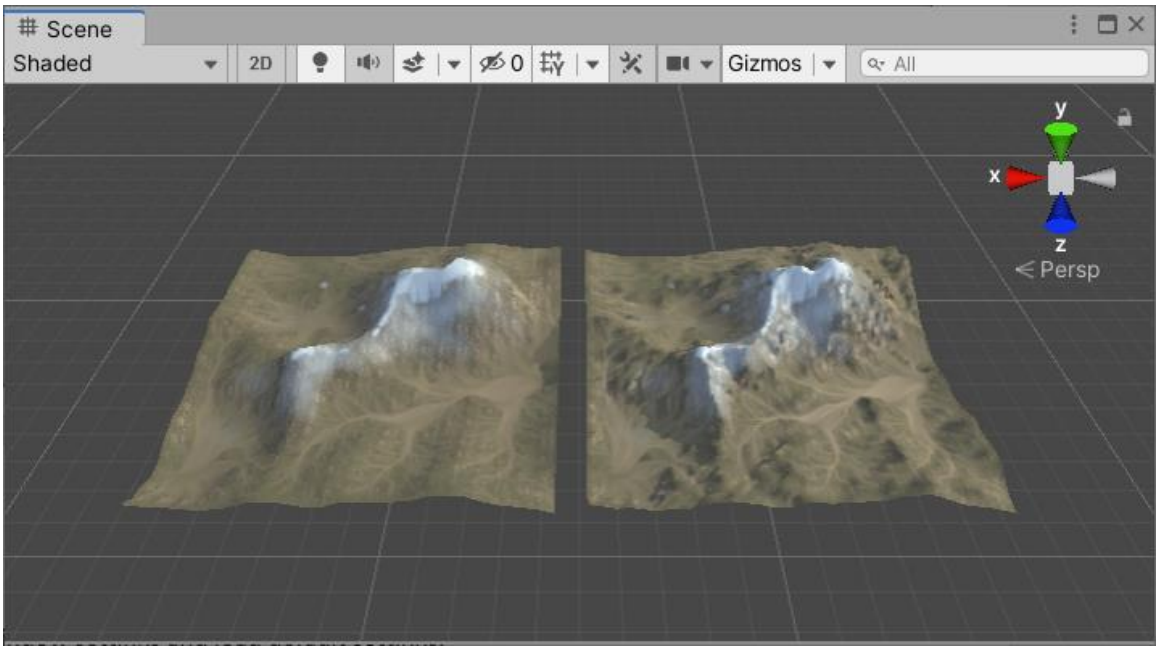


Add scene terrain object (SnowTerrain) into TTM window by drag & drop it from the Hierarchy window into the TTM editor window or by clicking on the **Add All Scene Terrains** button.

TTM window will display terrain and its resource usage.



Click on the **Run** button.
TTM will convert terrain (SnowTerrain) into a mesh, create material for it using Splatmap shader and instantiate ready to use prefab in the scene in the same position as the source Unity terrain object.

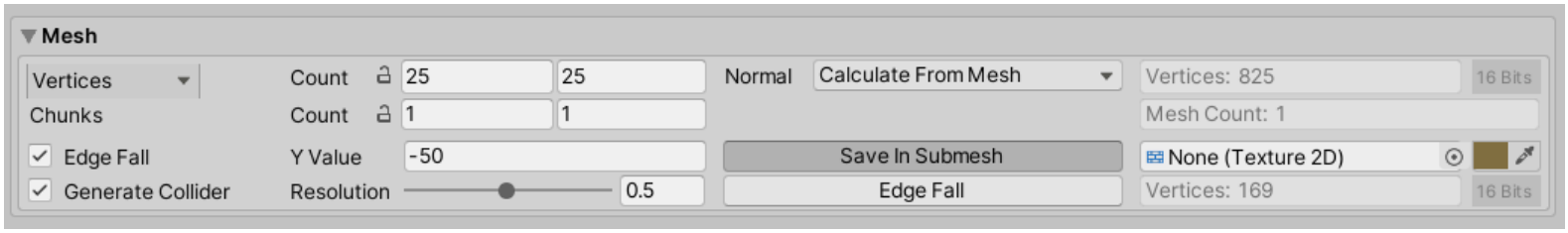


(To see generated mesh, move it inside Scene view or hide source Unity terrain object)

Note, if Unity Console window displays Splatmap shader compilation errors, check Update Splatmap Shaders chapter bellow.

EDITOR WINDOW SETTINGS

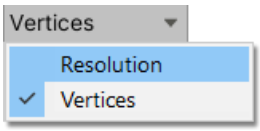
MESH



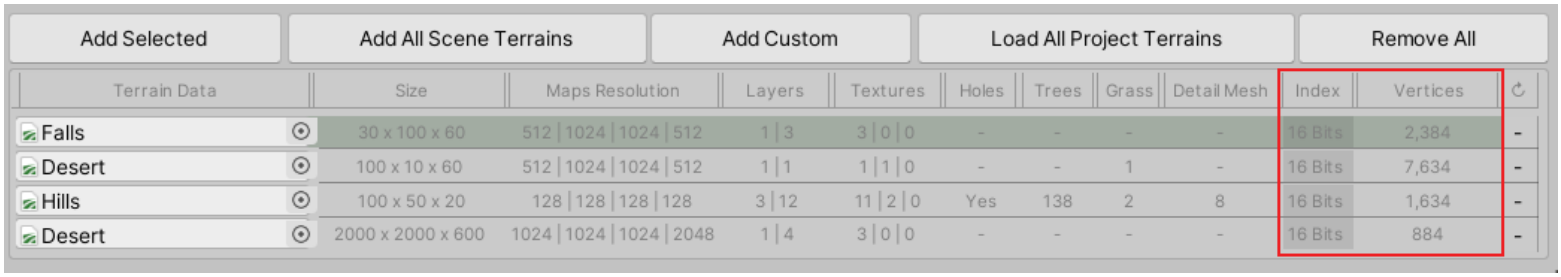
- **Vertices** – Controls generated mesh vertex count horizontally and vertically:

Final vertex count is displayed in the upper right corner: 

Depending on the vertex count TTM generates 16 or 32 bits index format meshes (more info).



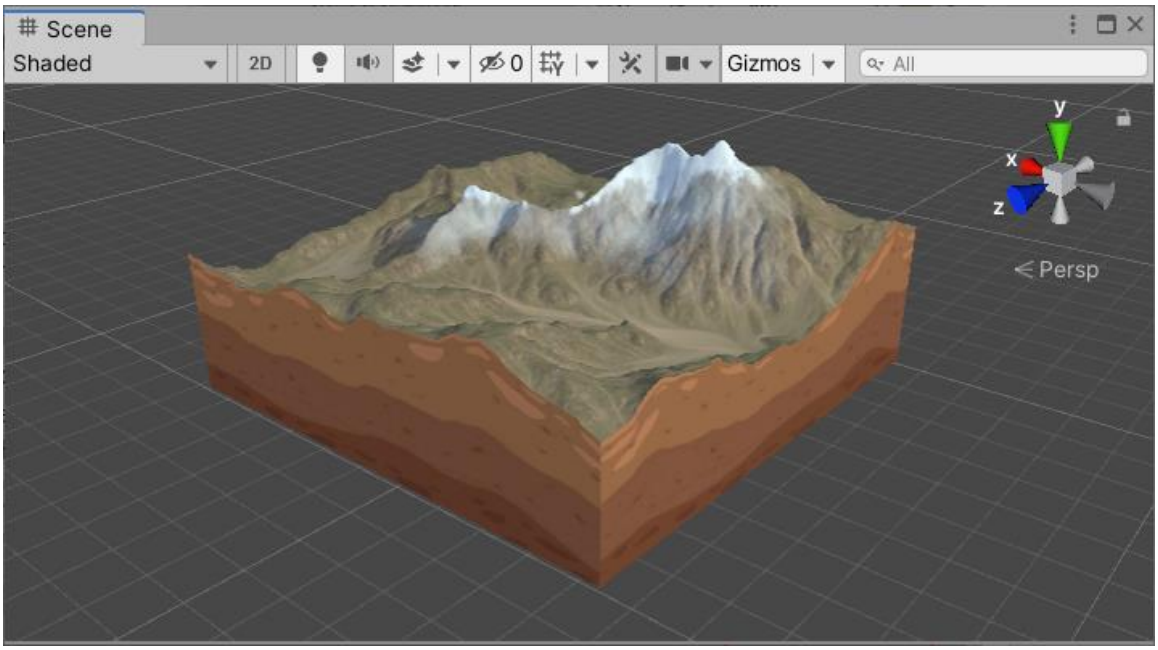
Choosing **Resolution** option instead of the **Vertices**, calculates vertex count horizontally and vertically in the way that vertex 2D grid always has quad shape. In this case final mesh vertex count depends on the source terrain length & width sizes and is displayed in the terrains list section.



> Note, TTM does not generate mesh with holes.
> Holes are supported as Alpha Cutout effect using shaders – the same way as it is done by Unity terrain system.

- **Chunks** – Splits source terrain into 2D grid and after that each part is converted into a mesh. Count property defines Horizontal and Vertical split amount.

- **Edge Fall** – Extrudes edges on the perimeter.



Y Value defines world space Y position for extruded vertices.

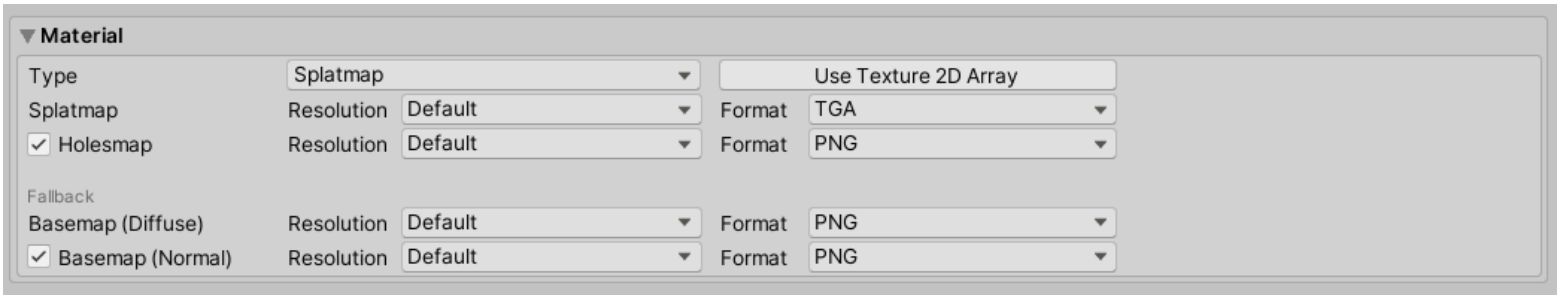
If **Save In Submesh** is disabled, extruded vertices will have same UV values as on the perimeter. If enabled, extruded vertices will be saved as a sub-mesh with new UVs and TTM will generate new material for them with simple procedural or user defined texture.

- **Generate Collider** – Creates separate mesh for collider use. Vertex count is calculating by (main mesh vertex count * **Resolution** value).

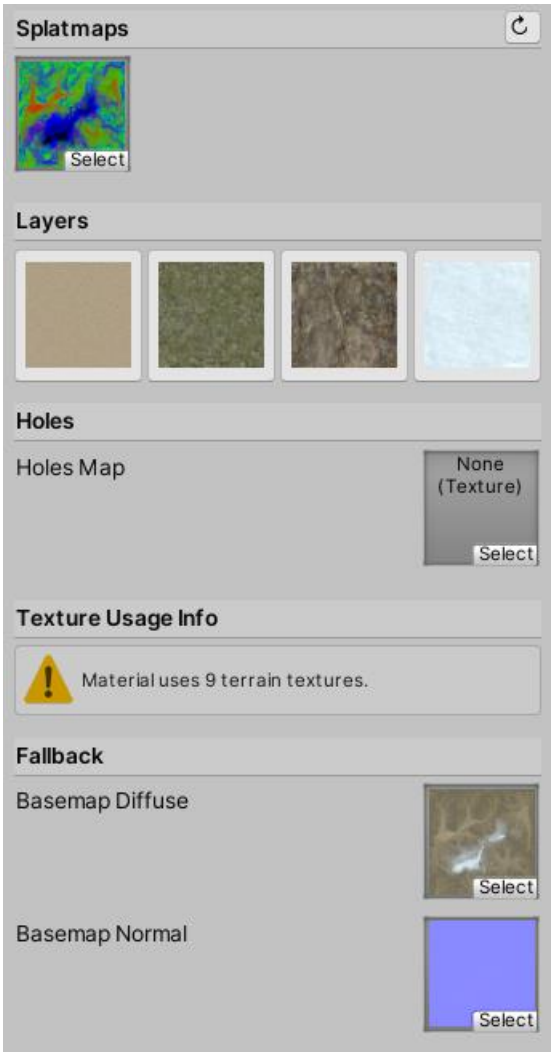
MATERIAL

In this group can be chosen material type for generated mesh and extracted paint textures from the source Unity terrain.

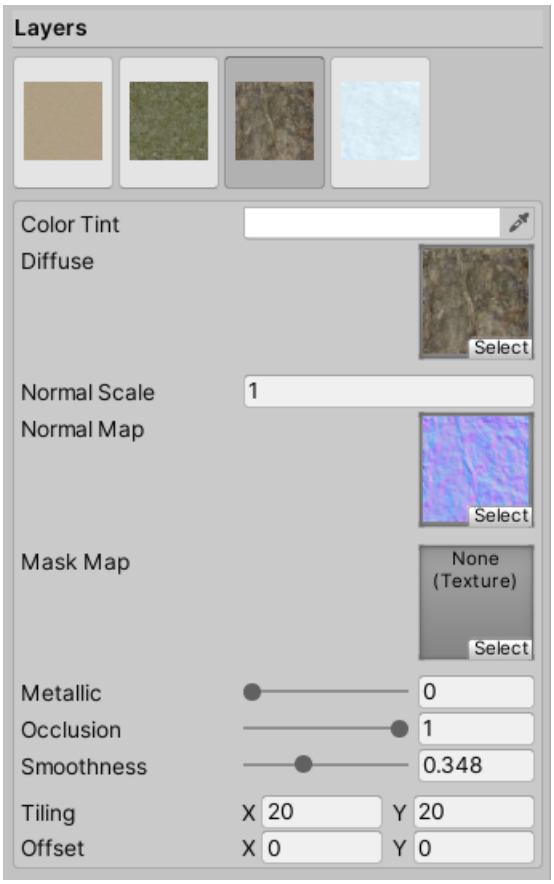
Note, TTM reads textures and material data from TerrainData object, not Terrain component and assumes source terrain uses Unity built-in shader.
TTM cannot export textures and materials from terrain using custom shader.



- **Splatmap** – Imitates Unity built-in terrain shader and can blend maximum 16 layers using 4 control maps - Splatmaps.



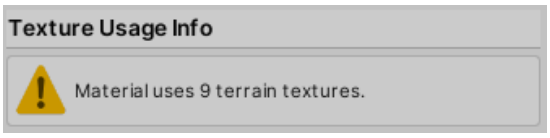
Each layer uses similar properties as Unity terrain layer.



When generating **Splatmap** material, TTM extracts splatmap and holesmap from source terrain and saves them in the same folder as the main mesh and prefab.
Diffuse, Normal and Mask map properties use source terrain layer resources.

By default, when generating **Splatmap** material TTM creates Basemap Diffuse texture too and assigns it to this material. This texture is used by Fallback shader only, which Unity should use if the main shader fails to compile.

When using **Splatmap** material always pay attention to the **Textures Usage Info**:



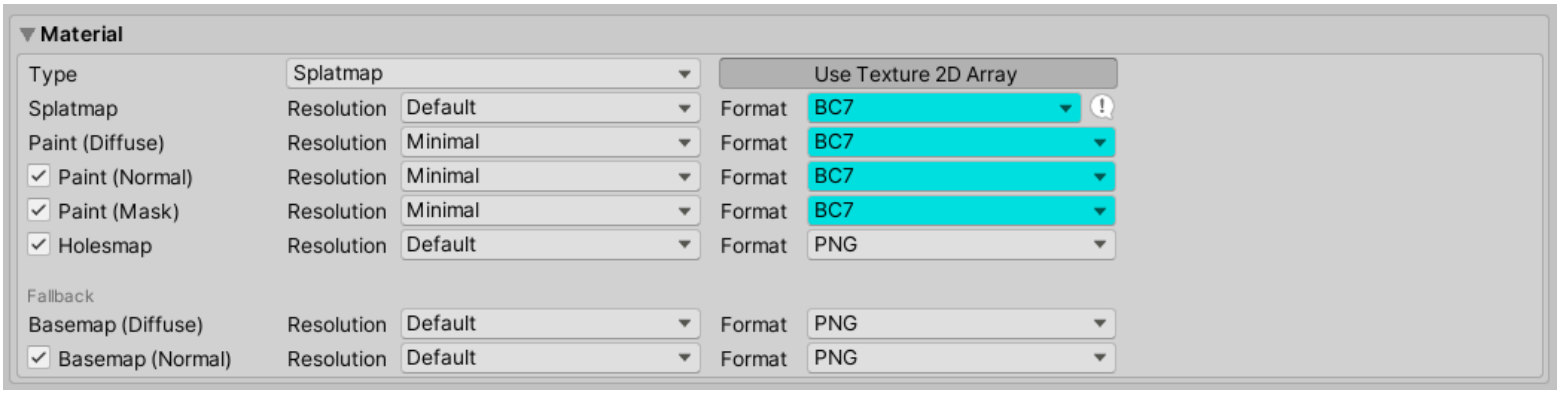
After building a project, some devices may fail to fully render **Splatmap** shader if material uses more textures than GPU supports, which varies from 6 to 8 textures for GLES2 mobile devices up to 16 for GLES3. DirectX 11 however can render 128 textures.

There are several scenarios if device cannot render **Splatmap** shader:

1. Mesh is rendered in complete 'pink' color. It means that device cannot compile shader at all.
2. Unity switches material to the Fallback shader and mesh is rendered using Basemap Diffuse texture created by TTM. This helps to avoid '*pink*' mesh rendering.
3. GPU renders as much textures as it can and just black color instead of all other textures.
4. Mesh is not rendered at all.

Always test **Splatmap** material on the oldest possible device.

If device does not support required amount of textures, TTM can bake them into Texture2DArray:

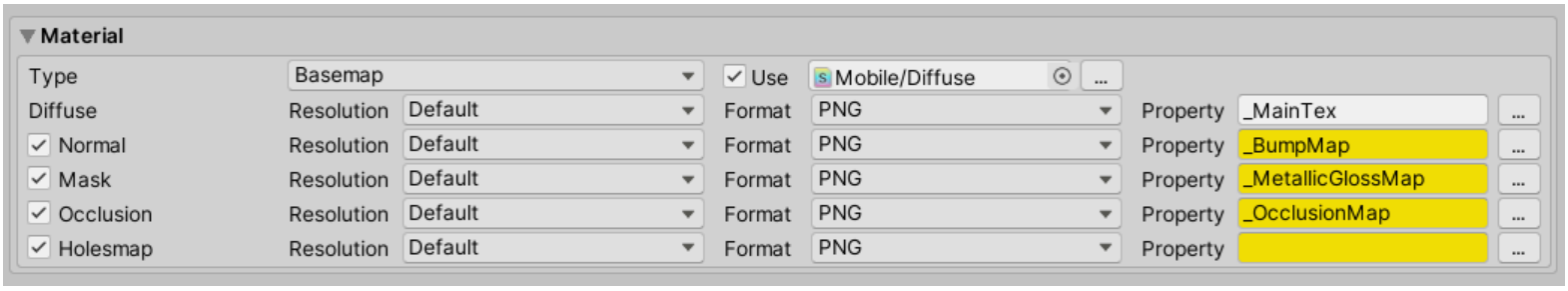


Each group of textures is baked in its own separate Texture2DArray file: Splatmap, Paint Diffuse, Paint Normal and Paint Mask.
Holesmap texture remains common texture 2D file.

Make sure target device supports Texture2DArray and used texture Formats using SystemInfo.

- **Basemap** – Bakes all terrain paint textures into one texture file and generates material with Unity default Standard/Lit shader.

  If **Use Custom Shader** is enabled, material will use any selected shader and baked textures will be assigned to the properties whose names are defined by **Property** fields.



Note, Property names not supported by selected shader have yellow background and baked textures will not be used in the generated material.

Note, Some shaders may require enabling appropriate keywords too for the assigned texture to have an effect. For example, Standard material uses NormalMap texture only if it has **_NORMALMAP** keyword enabled, or **_METALLICGLOSSMAP** keyword for using Metallic texture.
TTM automatically enables those keywords. But if **Use Custom Shader** is enabled, they must be managed manually (by default through material editor).

Compared to the **Splatmap** material, **Basemap** can be used on any device with any shader. But baked textures are limited to 8K resolution per-chunk.
Splatmap material does not bake anything and uses terrain paint textures in original resolution, but layers count is limit to 16 and used textures count depends on a device GPU.

Note, When using **Basemap** material, Holesmap can be baked inside Diffuse texture's Alpha channel:

OBJECTS

Allows extracting additional terrain resources.

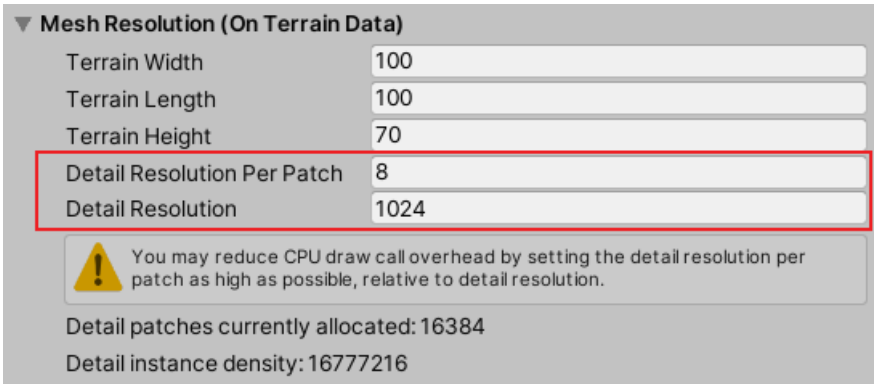
- **Trees** – Exports trees.

Exported objects are original tree prefabs and do not have terrain tree rendering features like billboard or distance fading, unless those features are already implemented into the prefab.
Rotation – Applies random or specific rotation angle to the exported tree.
Slope – By default exported trees Up vector is always oriented along (0, 1, 0) vector. This value rotates Up vector to be oriented along surface Normal.

- **Grass** – Exports terrain grass.

Each exported grass is a quad mesh and uses Unity built-in Mobile Diffuse material with texture used by this grass inside terrain system.
Exported mesh does not have terrain grass rendering features like billboard, distance fading, wind, etc.

Unity terrain system does not store each grass position, instead this data is saved inside 2D grid whose Row/Column size is defined by **Detail Resolution** inside Unity terrain settings.

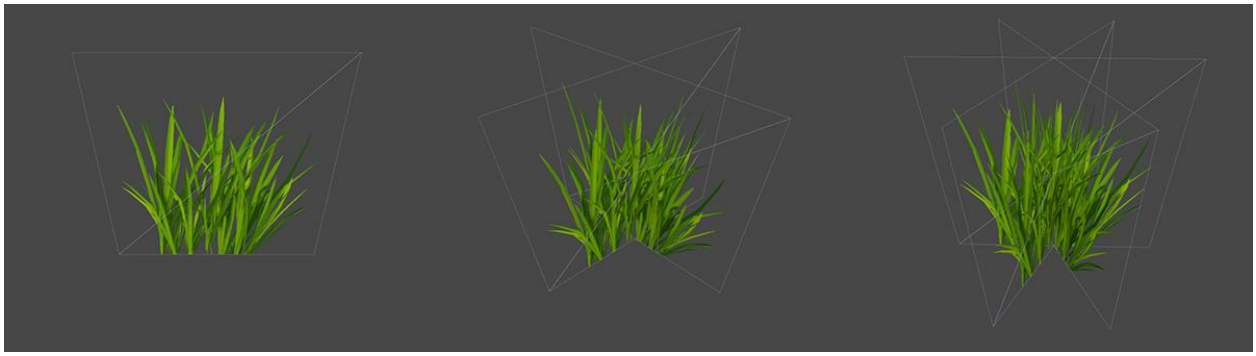


With higher **Detail Resolution** Unity allows to place grass on a terrain with greater accuracy.
Amount of generated grass in each cell inside this 2D grid is defined by **Details Resolution Per-Patch** option.

Per Patch option inside TTM editor window allows to minimize generated grass mesh count per patch.

Multiplier – Exported grass mesh count multiplier.

Sides – By default generated grass mesh uses quad mesh with one side. This option allows using mesh with multiple sides.



Rotation – Applies rotation to the exported grass mesh.
Slope – Controls whether grass mesh orientation follows mesh surface normal. By default it is always oriented in (0, 1, 0) direction – up vector.

Combine – Combines generated meshes.

1. **By texture** – TTM combines grass meshes by used textures. If desired Mesh Index format is set to 16 bits and combined mesh vertex count exceeds 65535, mesh is automatically split.

   Note, in the case of using 32 bit meshes, make sure target device supports them.

2. **Create Atlas Texture** – Instead of rendering each grass mesh separately by its own material, TTM combines used grass textures into one Atlas file and creates one material using this texture. All grass meshes use this one material.
3. **Everything** – Combines all meshes into one mesh, all textures into one Atlas file and one material is used for grass rendering.

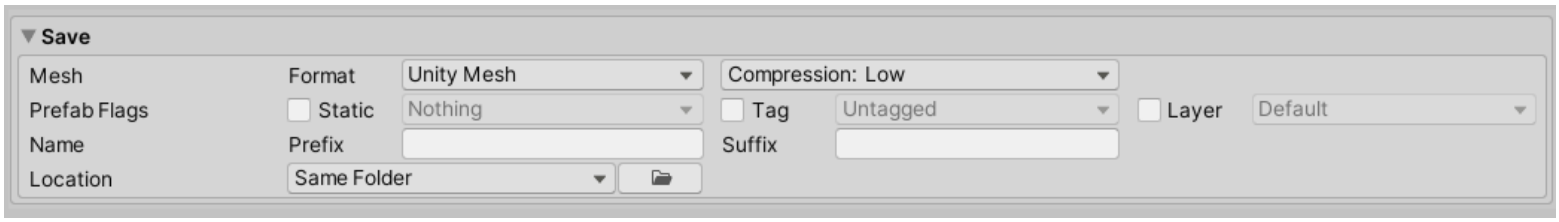
TTM asset package does not include any special grass shaders. Generated grass mesh is rendered using Unity built-in Mobile Diffuse shader. In the case of using any custom grass shader, TTM bakes additional data inside generated mesh:

- **Vertex Color RGB** channel contains grass Healthy & Dry color values used by this grass inside Unity terrain system.
- **Vertex Color Alpha** channel contains grass quad mesh's Top & Bottom values.
- Mesh UV4 (texcoord3 for shader) contains quad mesh pivot point position. Can be used in billboard shader.

- **Detail Mesh** – Exports terrain detail meshes.

Exported objects are original detail mesh prefabs and do not have terrain rendering features like billboard or distance fading, unless those features are already implemented into the prefab.

SAVE



- **Format** – TTM can save generated mesh in Unity ***.asset*** or ***.OBJ*** file formats.

Note, OBJ format does not support mesh vertex color.

Mesh **Compression** setting allows reducing generated file size by lowering numerical accuracy of the mesh. Instead of 32-bit floats, lower size fixed number will be used to represent mesh data.

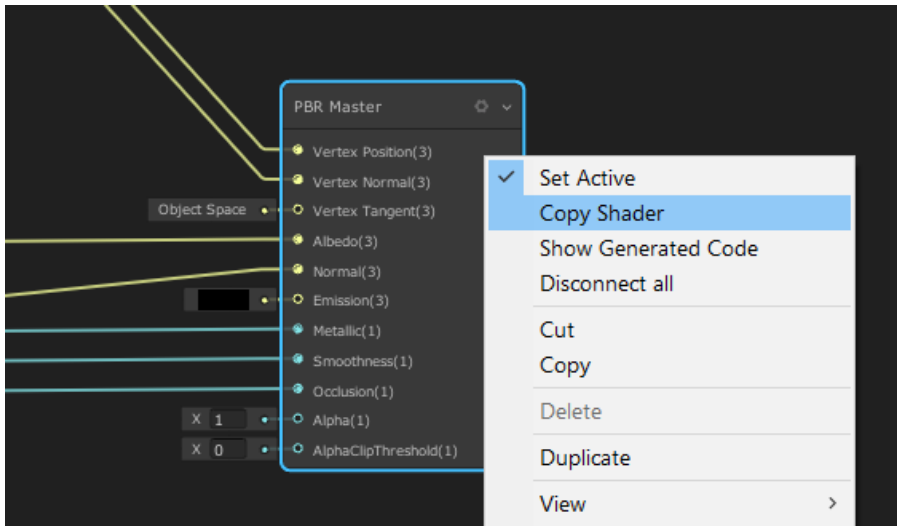
Note, More compression introduces more artifacts in the vertex data (*position*, *normal*, *uv*). After creating high compressed meshes, they may have visible seams on the edges and perimeter that are automatically fixed in game mode by **TerrainToMeshConversionDetails** script attached to the main prefab.

- **Prefab Flags** – Assigns Unity object flags to the generated prefab.
- **Name** – Adds prefix/suffix to all file names generated by TTM converter.
- **Location** – Generated files save location.

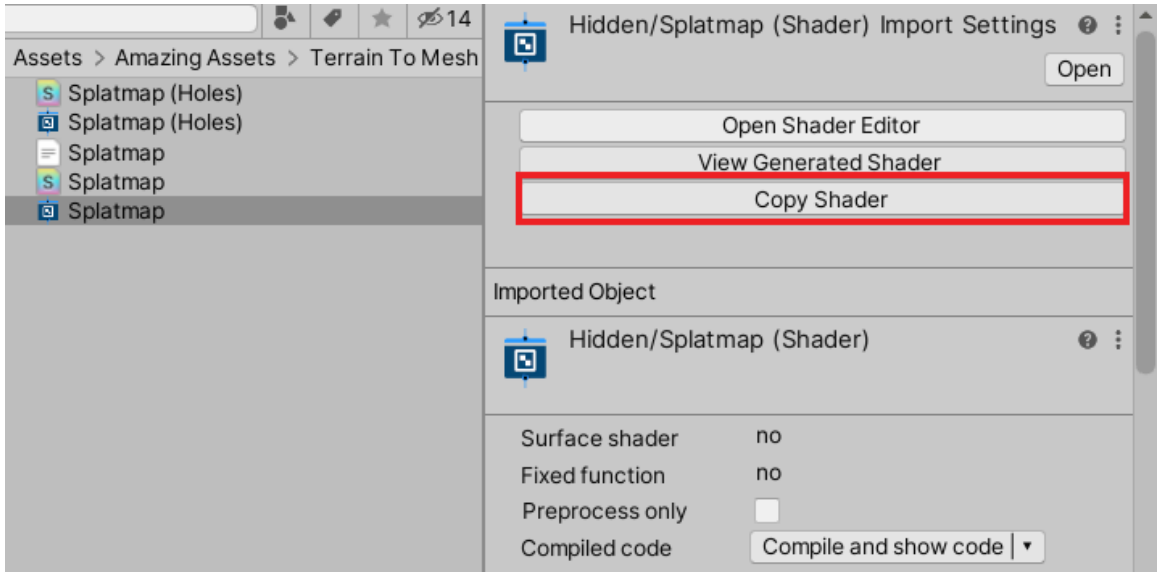
UPDATE SPLATMAP SHADERS

Depending on the Unity Editor version and used Universal or High Definition render pipeline versions, package included Splatmap shaders may require to be recompiled. This process is not automated and must be performed manually.

1. Copy shader HLSL code:
   (For Unity 2019.4) Open Splatmap *.shadergraph* file from the **Terrain To Mesh \ Shaders \ Splatmap** folder and using context menu on the PBR Master node, select Copy Shader option.

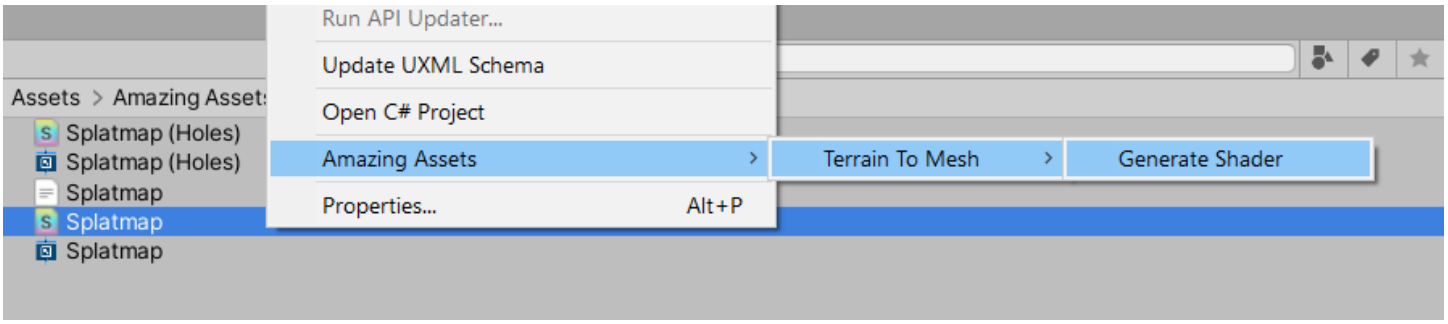


   (For Unity 2020 and later versions) Select Splatmap *.shadergraph* file inside **Terrain To Mesh \ Shaders \ Splatmap** folder and inside Inspector window click on the Copy Shader button.



2. Generate shader:
   After copying *.shadergraph* code, select Splatmap *.shader* file in the same directory and from the context menu choose **Amazing Assets \ Terrain To Mesh \Generate Shader** option.

   This will recompile shader using currently installed SRP package.



3. Repeat steps 1 and 2 for the **Splatmap (Holes)** shader too.

Note, Unity Console window may display warnings about “implicit truncation of vector type” inside shader, ignore them.

RUN-TIME API

Terrain To Mesh extension methods can be brought into scope with this using directive:

C#
```
using AmazingAssets.TerrainToMesh;
```

Unity TerrainData class now will have **TerrainToMesh()** extension with following methods:

```
public Mesh ExportMesh (int vertexCountHorizontal, int vertexCountVertical,
                        Normal normalReconstruction, EdgeFall edgeFall = null)
```

Exports mesh from TerrainData object.

`normalReconstruction` – Calculates mesh *Normal* from source terrain or using mesh after it is generated.

`edgeFall` – Options for generating edge fall. Not used by default.

public class EdgeFall
```
public float yValue;
public bool saveInSubmesh;
```

```
public Mesh[]  ExportMesh (int vertexCountHorizontal, int vertexCountVertical,
                           int chunkCountHorizontal, int chunkCountVertical,
                           bool perChunkUV, Normal normalReconstruction, EdgeFall edgeFall = null)
```

Splits terrain into 2D grid and exports it as mesh array. Vertex count is defined per-chunk.

`perChunkUV` – If enabled, each mesh has UVs in the range of [0, 1].

```
public Mesh  ExportMesh (int vertexCountHorizontal, int vertexCountVertical,
                         int chunkCountHorizontal, int chunkCountVertical,
                         int positionX, int positionY,
                         bool perChunkUV, Normal normalReconstruction, EdgeFall edgeFall = null)
```

Splits terrain into 2D grid and exports one mesh from this array based on **`positionX`** and **`positionY`** values.

```
public Texture2D ExportHolesmapTexture (int resolution, bool unpack)
```

Exports holesmap texture.

resolution – Exported texture resolution in the range of [16, 8192].
unpack – Unpacks texture for saving it into a file (for editor use). If texture after exporting is directly used in material, then this value must be false.

```
public Texture2D ExportBasemapDiffuseTexture (int resolution, bool includeHolesmap, bool unpack)
```

Exports basemap texture - Diffuse.
includeHolesmap – If enabled, exported texture's alpha channel contains holesmap value.

```
public Texture2D ExportBasemapNormalTexture (int resolution, bool unpack)
```

Exports basemap texture - Normalmap.

```
public Texture2D ExportBasemapMaskTexture (int resolution, bool unpack)
```

Exports basemap texture – Maskmap (Red channel contains Metallic, Green – Occlusion, Alpha - Smoothness).

```
public Texture2D ExportBasemapOcclusionTexture (int resolution, bool unpack)
```

Exports basemap texture – Occlusion.

All textures provided above can be exported as 2D grid, exactly the same way as meshes.

```
public Texture2D[] #Texture Export Method# (int resolution,
                                            int chunkCountHorizontal, int chunkCountVertical,
                                            bool unpack)
```

```
public Texture2D #Texture Export Method# (int resolution,
                                          int chunkCountHorizontal, int chunkCountVertical,
                                          int positionX, int positionY, bool unpack)
```

```
public TerrainLayer[] ExportTerrainLayers ()
```

Exports TerrainLayers.

```
public Texture2D[] ExportSplatmapTextures(int resolution, bool unpack)
```

Exports splatmap textures used blending paint textures.

resolution – Exported texture resolution in the range of [16, 8192].
unpack – Unpacks texture for saving it into a file (for editor use). If texture after exporting is directly used in material, then this value must be false.

```
public Material ExportSplatmapMaterial (bool hasHolesmap)
```

Exports splatmap material imitating Unity built-in terrain shader, with maximum 16 layers support.
hasHolesmap – If enabled, splatmap material will use Alpha Cutout effect for creating holes based on holesmap value.

Note, For run-time texture export always include AllTerrainTextures.shader from **Terrain To Mesh \ Shaders \ All Terrain Textures** folder into the build.

```
public TreePrototypesData[] ExportTreeData()
```

Exports tree prototypes data from TerrainData object.

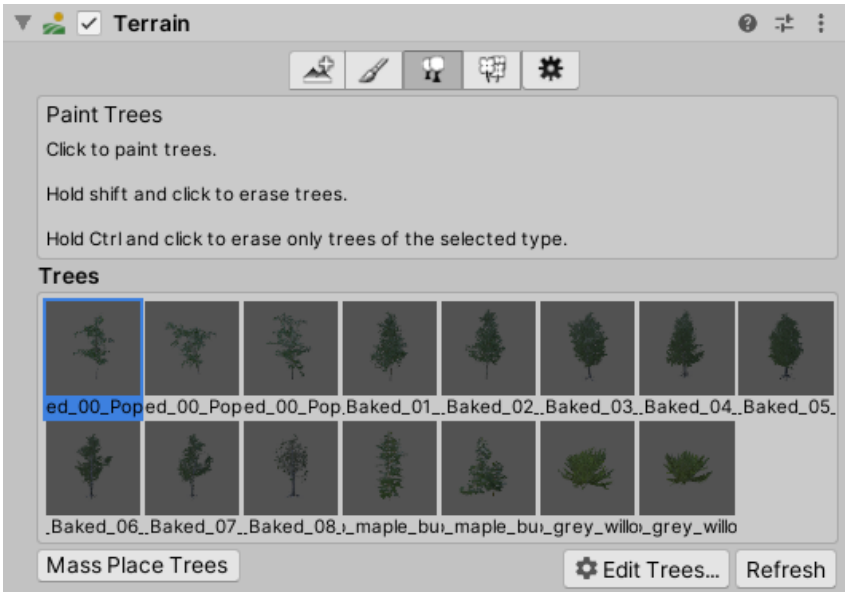
```
public class TreePrototypesData
public GameObject prefab;

public int prototypeIndex;

public List<Vector3> position;
public List<Vector3> surfaceNormal;
public List<Vector3> scale;
```

prefab – Original tree prefab object.

prototypeIndex – Index of a tree prefab in the TerrainData.treePrototypes array.



position – List off all positions (in terrain space, not world space) where this tree prefab is used.

surfaceNormal – Surface *Normal* direction at *position*.

scale – Tree object scale at *position*.

```
public DetailPrototypesData[] ExportGrassData (int maxCountPerPatch, float countMulitplier)
```

```
public DetailPrototypesData[] ExportDetailMeshData (int maxCountPerPatch, float countMulitplier)
```

Exports grass and detail mesh data from TerrainData object.

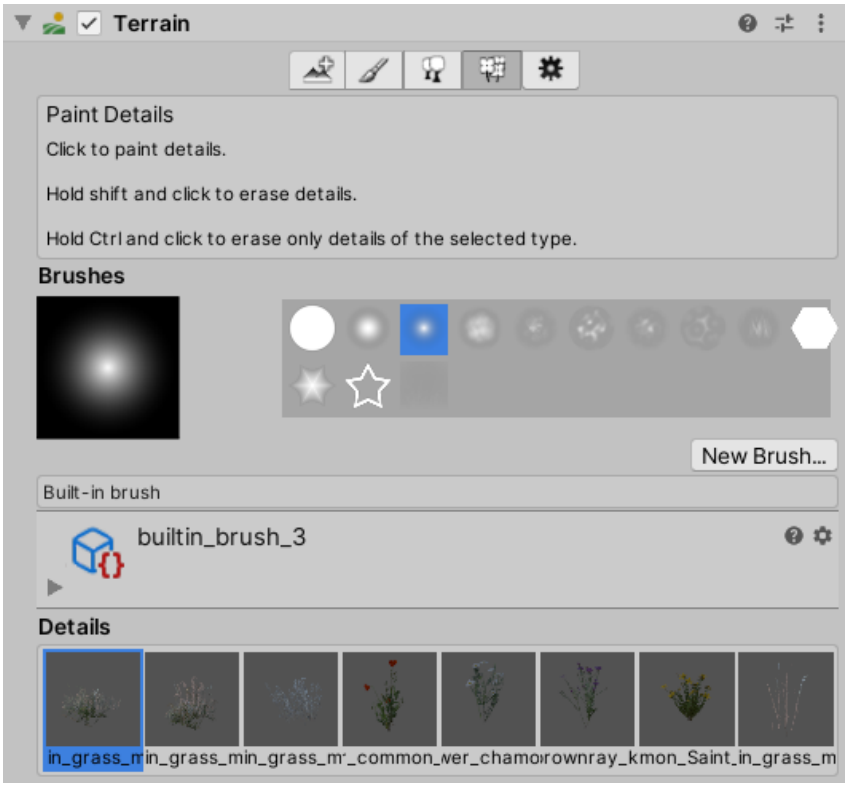
public class DetailPrototypesData

```
public DetailPrototype detailPrototype;

public int prototypeIndex;

public List<Vector3> position;
public List<Vector3> surfaceNormal;
public List<Vector3> scale;
public List<Color> healthDryColor;
```

detailPrototype – Unity DetailPrototype object used by grass or detail mesh.
prototypeIndex – Index of a detail prefab in the TerrainData. detailPrototypes array.



position – List off all positions (in terrain space, not world space) where this detailPrototype is used.
surfaceNormal – Surface *Normal* direction at *position*.
scale – DetailPrototype's scale at *position*.
healthDryColor – DetailPrototype's healthy & dry color at *position.*

Using AmazingAssets.TerrainToMesh namespace adds **Utilities** class with following methods:

```
public static string ConvertMeshToOBJ(Mesh mesh)
```

Converts mesh to OBJ format string.

```
public static void ConvertMeshToOBJ(Mesh mesh, StreamWriter streamWriter)
```

Saves mesh in OBJ format file using **streamWriter**.

```
static public int GetGeneratedVertexCount(int vertexCountHorizontal, int vertexCountVertical,
                                          bool hasEdgeFall)
```

Calculates vertex count for mesh converted from terrain.

```
static public bool HasHoles(TerrainData terrainData)
```

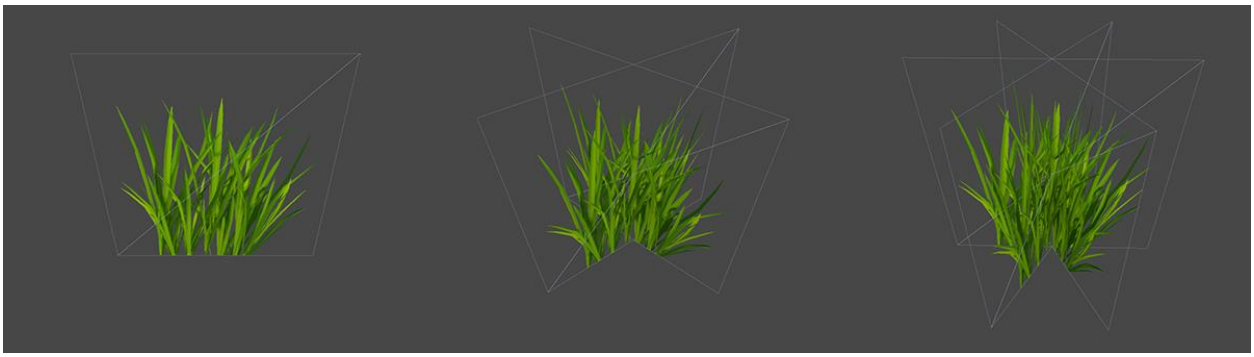
Checks if terrain has holes.

```
static public bool HasTextureAlphaChannel(Texture2D texture)
```

Checks if texture has alpha channel.

```
static public Mesh CreateGrassMesh(int sides)
```

Creates quad mesh for grass rendering.
sides – Cross section count in the range of [1, 6].



Vertex color's alpha channel contains quad mesh Top & Bottom values: Bottom – 0, top – 1.
Pivot point is in (0, 0, 0) position.

```
static public List<Mesh> CombineGameObjects(GameObject parentGameObject, Material material,
                                            string meshName, string objectName,
                                            UnityEngine.Rendering.IndexFormat indexFormat)
```

Combines all child meshes under **parentGameObject** object into one mesh and renders it using **material**. If **indexFormat** is 16 bit and combined mesh requires more than 65535 vertices, it is automatically split. Returns list of the generated combined meshes.

```
static public void ConvertResolutionToVertexCount(TerrainData terrainData, int resolution,
                                  out int vertexCountHorizontal, out int vertexCountVertical)
```

Calculates vertex horizontal and vertical counts when converting terrain into a mesh using **resolution** value.