Back Story

Hi and thank you for purchasing this mesh generation tool.

The main objective of this collection of scripts is to
1. Allow you to easily generate meshes on the fly with several types of UV sets.
2. To provide you with reusable / easy to understand methods so that you can adjust or expand the toolkit as you see fit.

Given these objectives, it does mean that these scripts are not necessarily the fastest ones out there, to generate their target shape, but instead are easier to dissect.

Usage

Each script has one or several *"Generate"* functions which will return the desired Mesh.
Example: *var mesh = CircleGenerator.Generate(0.5f, Vector3.zero);*

All scripts provided can be used while in or out of Unity's "play mode"

Important. All meshes generated are stored in the generator itself and override one another. So when generating a cylinder (or any other shape) the CylinderGenerator has a private Mesh variable which is re-used instead of making an "new Mesh()" everytime. This means that when calling the cylinder generator a second time the first cylinder will also change as they are both working on the same variable.

"So how to generate multiple different cylinders?"

The answer is either.
1. Use .**Clone()** on the generated mesh, this will duplicate the mesh in a new Mesh, removing any links to the Generator.
2. Use **CombineMeshes.Combine()** to merge an existing mesh reference with the generated mesh. This will likely be the preferred method in most situations, to avoid Unity's tendency to fill memory with Mesh references.

Please check out the **Demo scene** which has multiple examples of individual and combined meshes.

If you are unaware of why to avoid constant use of "new Mesh()". I strongly encourage you to checkout Nothke's blog titled: "`Procedurally generated meshes in Unity without garbage`"

http://nothkedev.blogspot.com/2018/08/procedurally-generated-meshes-in-unity.html

Notes

- The final vert count can sometimes be a little different then expected.

  In most cases the model is made in a way to support hard edges, meaning each face has its own set of verts, uvs, normals etc. For example a cube does not have only 8 vertices, but instead has 4 (corners) * 6 (sides) = 24 vertices.

  Similarly a circle with a resolution of 20 actually has 21 vertices as the first and final vert are placed on the same spot to close the circle and let the UVs properly flow between 0 - 1. This results in visually having 20 corners, matching the original input.

- Each Mesh generation script has a summary at the top detailing the data put in each of the UV channels, which can help for things like custom shaders or perhaps you want to swap some of the UV data around, to better support your materials.

- There is a “MeshTest” material which has 2 sliders, allowing you to look at the UV mapping in the other channels. This material is used in the Demo scene.

- The Demo folder can be removed without loss of functionality

Author

Glenn Korver