Procedural Sprite Generation v.1.0

For Unity

Hello there! Thanks for downloading this package.

In this brief guide, I'll give you quick overview about PSG.

Why did I made it?

In Unity there's no system enabling developer to easily create sprites directly from the code. Except from generic mesh generation, programmer must create all shapes and sprites manually – so why don't give an opportunity to facilitate it?

Once upon a time, I faced the problem of creating simple car based on sprites – moreover, I wanted them to be random, and there was no such thing in Unity. So I created PSG.

How to use it?

To add a PSG sprite from code, you need to do several steps:

- In arbitrary script, just add empty *GameObject* and then attach any of *MeshBase*-derived script to it.
- Then, use *Build()* function on it with corresponding parameters to create desired shape. Don't worry about adding renderers or colliders, the script will address it.
- You mesh is ready, now just hit "Play" or add more components to your object, just like *Rigidbody2D* or any *Joint2D*.

To explore scripting even deeper, take a look into the included sample scenes. There are plenty of examples, so feel free to explore them.

If you want to use dynamic lighting, you need to attach proper material to your shapes. Otherwise, the best choice is build-in *Sprites-Default*

Warning #1: Adding *MeshBase*-derived script to *GameObject* while in Inspector does not create the mesh! Those scripts act more like guidelines for building the mesh from script.

Warning #2: If you want to get mesh components (vertices, triangles, etc.), get them directly by *MeshFilter* component. If you write your own getters for *MeshBase* classes' vertices or triangles, they may not be the current if other script has been modifying them.

How it works?

Everything in Unity – 3D models, sprites, particles – is a mesh. A mesh consists of vertices, triangles, UVs and normals. Vertices are basically the points in space, representing model's vertices, and triangles make the faces of the mesh – all shapes can be represented by a set of triangles.

So what about the UVs? Their main purpose is to enable the model texturing. Bind to each vertex, they represent fractions of a texture on the model, so your rendering engine knows how to display the textures.

Normals are used for lighting and display. They are basically directional vectors, one associated with each vertex. In PSG, we use sprites, so normals are important only when using physics-based lighting. In PSG, all meshes are flat, because sprites live in 2D environment.

When building any kind of mesh, first vertices and triangles are added to *Mesh*. Then, sprite is UV unwrapped and normals are added – in our case, all of them are just *Vector3.ups* ({0,1,0}). Finally, resulting mesh is attached to *MeshFilter* component.

While roaming Unity's scripting reference, you may bump into *Optimize* method in *Mesh* manual. What it does is optimize (what a surprise) *MeshFilter*'s mesh for rendering at the expense of loading time. However, the impact on loading isn't very big, so in PSG it's enabled by default. You can switch it on and off by changing the value of static field *OptimizeMesh* in *MeshBase*.

If you'd like to know more:

- Meshes: https://unity3d.com/learn/tutorials/topics/graphics/meshes
- Mesh::Optimize: https://docs.unity3d.com/ScriptReference/Mesh.Optimize.html

TODO (in future releases):

- Optimisation of Gears
- Scripts to add meshes directly from Inspector
- Ability to choose collider (Poly or Circle) of CircleMesh
- Collision groups
- Non-convex shapes (+triangulation)
- Shape smoothing

If you have any concerns or questions, feel free to send me an e-mail:
Wawrzyn321@gmail.com

PSG on Github:
https://github.com/Wawrzyn321/Procedural-Sprite-Generation

Best regards!