Procedural Sprite Generation Documentation
1.2.0

Wawrzyn321

1 About the project

The project goal is to make the procedural creation of 2D shapes in Unity easier. Currently there are 15 available shapes. Also, the use of generated shapes is faciliated by enabling the user to add various physical joints and modify other properties of shapes.
Examples scenes have been included into the package, I will briefly describe all of them below.

2 Example Scenes

2.1 Test scene

A little box with all shapes included in objects, hinges, fixed joints and physical interaction. You can zoom, pan and drag the objects with mouse (and also in all other scenes).

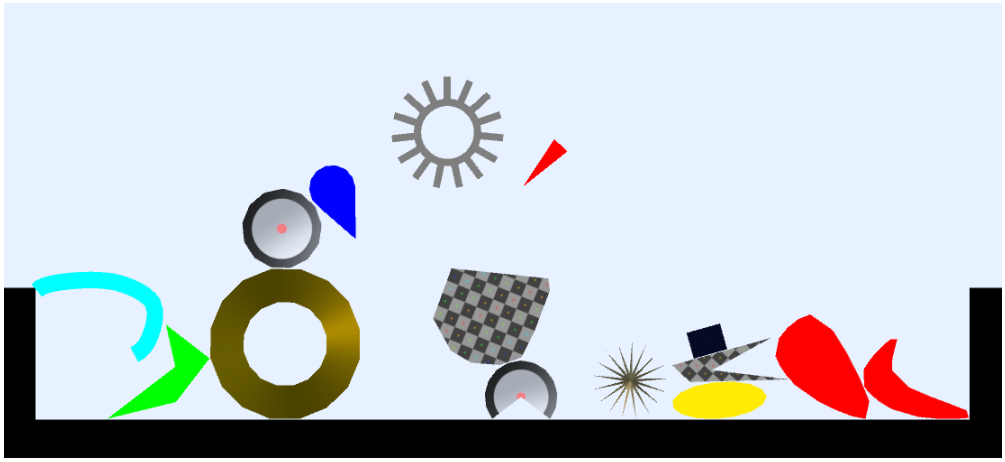

Figure 1: A little sandbox

2.2 MeshCreator

This time it's a scene for just for Editor, with main script - *MeshCreator*. You can choose a type of shape and properties - both common for all meshes (like physics) and specific for each mesh type.

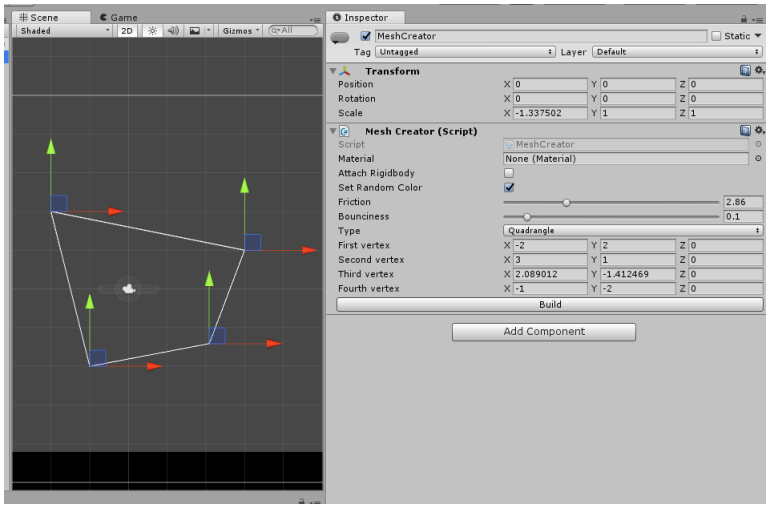

Figure 2: Here we are building a quadrangle

2.3 ConvexMesh

A couple of shapes are rotating, suspended on hinge joints, and the ConvexMesh is generated every Fixed Update.

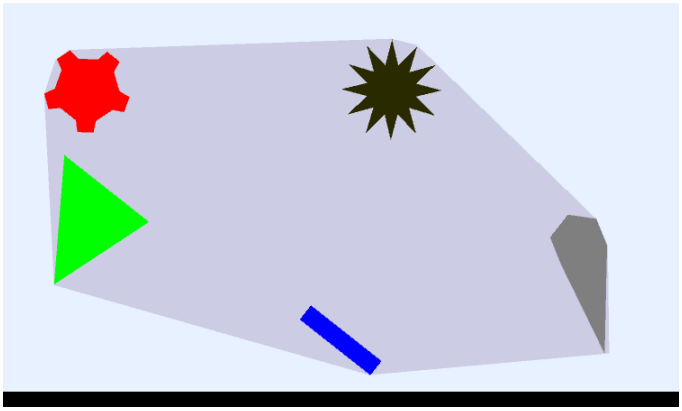

Figure 3: A pinnacle of ConvexMesh possibilities

2.4 Car

Simple scene with randomly generated "car" (just a quadrangle with two wheels). Nothing but only a test of random generation.

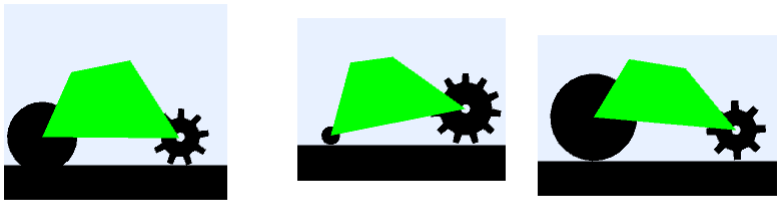

Figure 4: Those cars are very sophisticated

2.5 LineMesh & Gears

Both scenes present a randomly generated Line and a couple of connected gears.

3 MeshBase class

3.1 Data and Validation

3.1.1 Data

Basic shape data is kept here - vertices, triangles and uvs arrays and *Mesh* along with *MeshFilter* and *MeshRenderer*.
All private fields are adorned with *SerializeField* and *HideInInspector* attributes - first to make the data persistent (otherwise after reloading the scene or copying them they would disappear), second to hide them.
I cached mathematical constants here, used thorough the project.
GetVerticesInGlobalSpace method is also worth mentioning - it transforms shape's vertices to world space.

3.2 Validation

Last but not least - *Validate* boolean. Set to true by default, forces the building functions to check the correctness of provided parameters (like radii, number of sides etc.). When any of values doesn't conform with expected values, mesh build stops and appropriate error is raised. When unset, all parameters can be passed, but the results may be... ugly.

3.3 Building process

Building the mesh has been separated into five sections:

- Validation - if necessary, script checks for errors in parameters.
- Material check - as we don't want to apply null Material for new shape, so in that case replace it with a default one.
- Getting necessary components - overrided functions are often very similar, we need to get filter, renderer and colliders.
- Building mesh components - the most interesting part - calculating and assigning verts, tris and uvs.
- Updating other components - inform the *MeshFilter* and specific collider about newly-born shape.

3.4 Joint Physics and Materials

You can add and specify *HingeJoint2D* and *FixedJoint2D* or join two meshes with the latter, or set graphics and physics for a shape.

4 Persistence

4.1 Saving and loading

Meshes can be saved - both as content of MeshFilter only and as a full-fledged Prefab. Remember not to duplicate names of meshes!

4.2 Rebuild functionality

In case you save the shape as Prefab, and then delete saved assets, original shape may disappear. Fortunately, the "Rebuild" option exists - you can rebuild the mesh from serialized values from *MeshBase* script.

5 Shapes

5.1 LineMesh

Polygonal chain with adjustable width, built upon a set of points. A random one is generated in *LineMeshDemo* scene.
You can use two types of colliders - *PolygonCollider2D* for precise physics or *EdgeCollider2D* for simplified, but faster collision detection. The latter goes through the center of each segment.

5.2 ConvexMesh

Here the convex hull of a set of provided points is computed. The best example of ConvexMesh in action is *ConvexMeshDemo* scene.
I used QuickHull for fast and efficient computing.

5.3 CakeMesh

Despites the name, nothing fancy - just a fan of triangles in a (roughly) shape of circle. When the number of sides is equal to number of filled sides, we get just ordinary circle.

5.4 TriangulatedMesh

Mesh built by triangulation of a set of vertices. Triangulation is done by utilizing fairly simple ear clipping triangulation.

5.5 Spline mehes

Meshes that use spline creation algorithm.

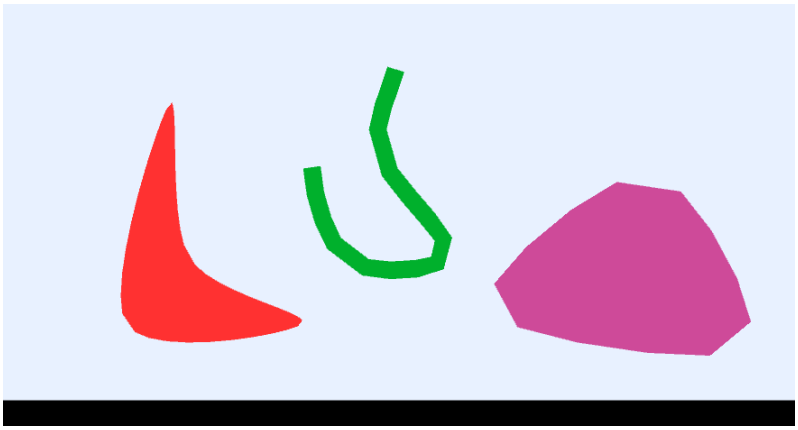

Figure 5: Spline shape, spline curve and convex spline shape

5.6 Other shapes

Shapes like:

- Triangle
- Circle
- Ellipse
- Gear
- Quadrangle
- Ring
- Star

- Pointed Circle

aren't described here - their structure and looks are straightforward.

Figure 6: The famous "Pointed Circle"

6 Contact

If you have any questions or issues, don't hesitate to contact me @ Wawrzyn321@gmail.com
Project's Github site: `https://github.com/Wawrzyn321/Procedural-Shape-Generation`