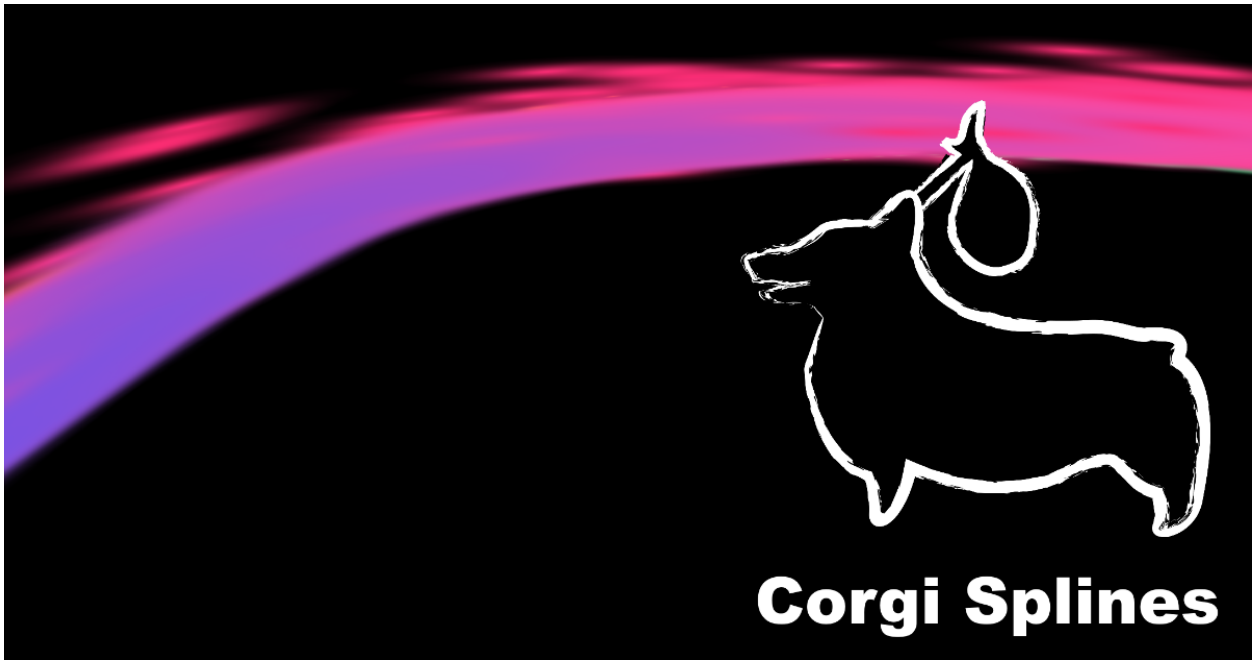


Corgi Splines Documentation (v1.5.1)

Thanks for purchasing Corgi Splines.
You can find script reference by unzipping the Docs.zip folder.
For help getting started, please keep reading.

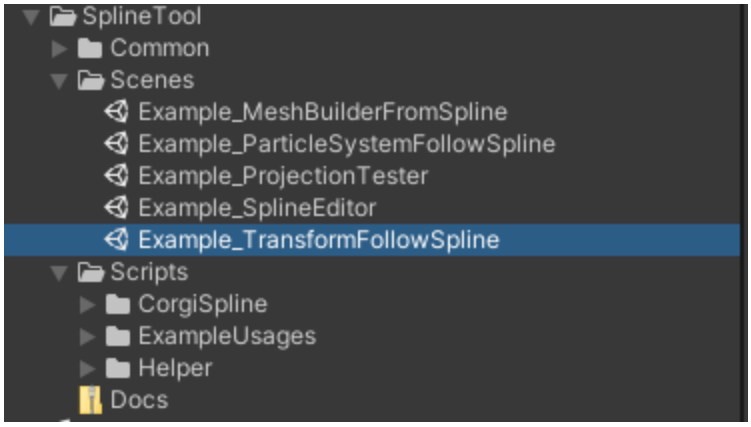
Getting started

If your project has errors, it's because Unity failed to automatically fetch the dependencies for you. You'll need the following packages installed:

- com.unity.burst
- com.unity.collections

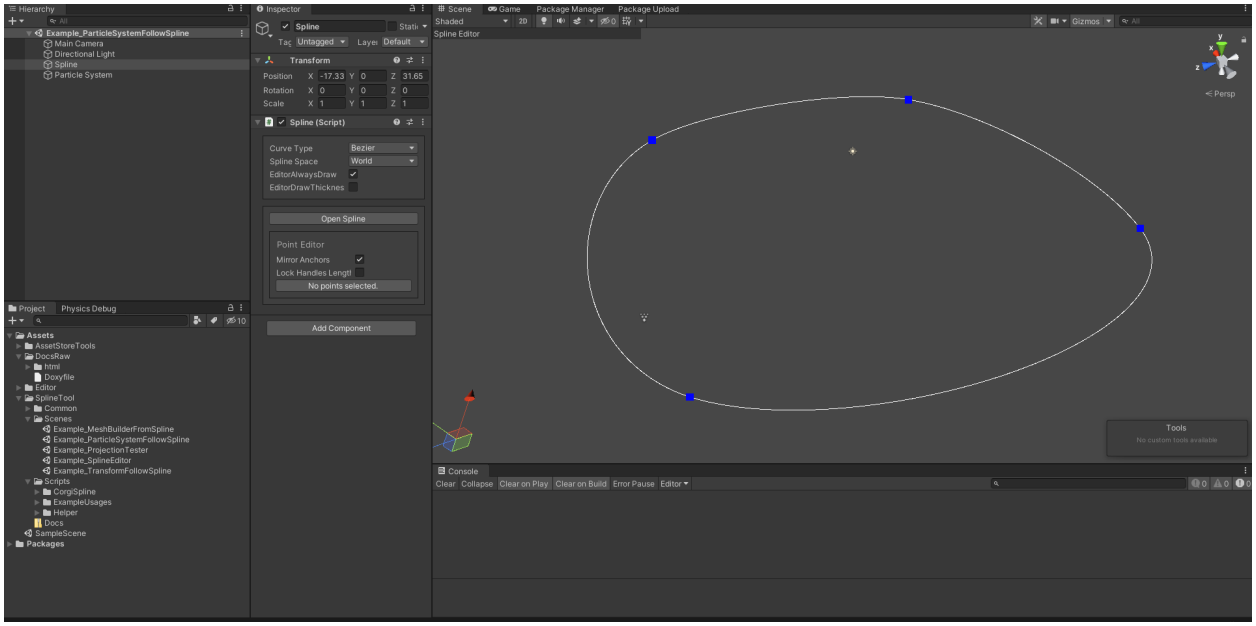
If they are not automatically installed, you can install them via the package manager by using the dropdown on the top left and typing in those package names.

When you import the project, you'll find the following folder structure. I've included a handful of example scenes, showing some uses of this spline tool.

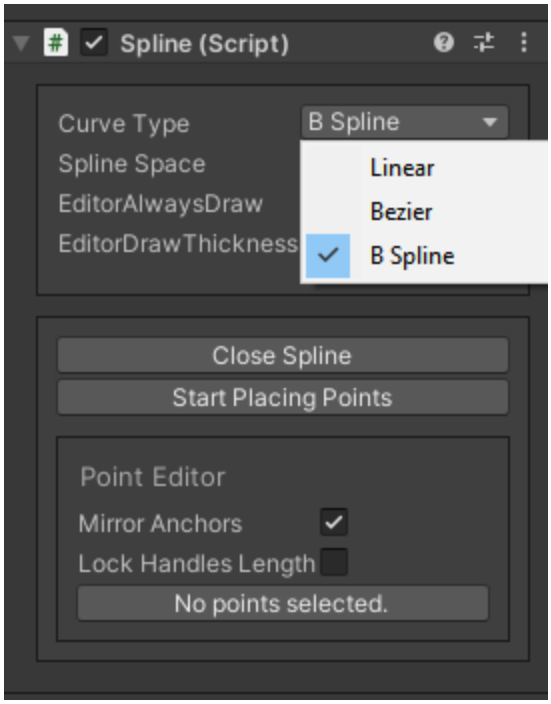


Example_SplineEditor.scene

This is probably the best place to start. Here you'll find the editor in a very simple scene, to see how everything works. To start, click on the **Spline** GameObject.



Spline Editor



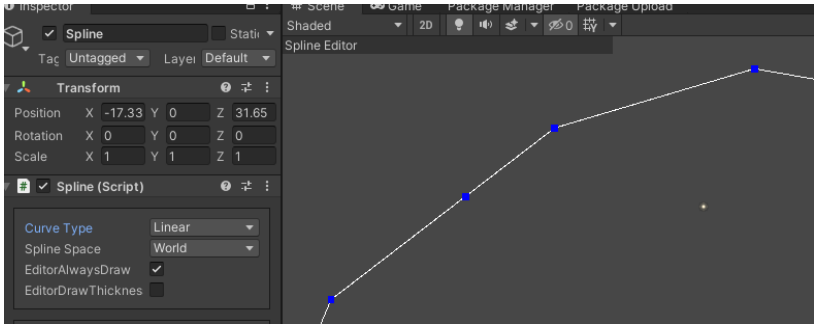
The Spline editor has two parts. The first box includes settings that are per-Spline. From top to bottom, we have Curve Type, Spline Space, EditorAlwaysDraw, and EditorDrawThickness.

Curve Type

Corgi Splines currently supports three types of curves: Linear, Bezier, and BSpline. Linear curves are what they sound like, when sampling from point to point, you'll draw a straight line. If you're looking for smoother curves, I recommend trying the Bezier and BSpline curve types.

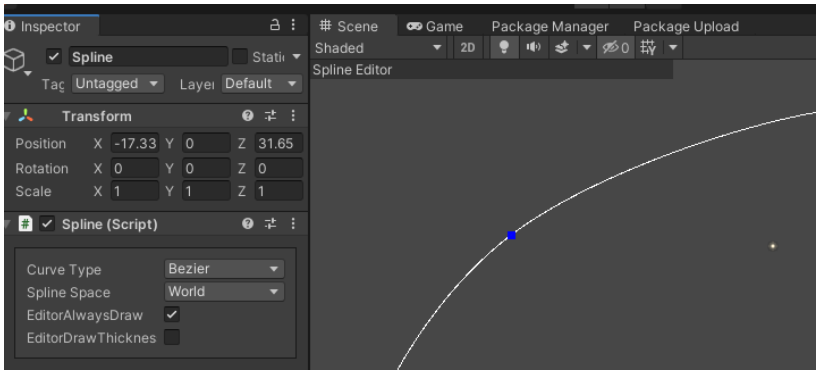
Linear

The spline is treated as a set of straight lines connecting the spline points.



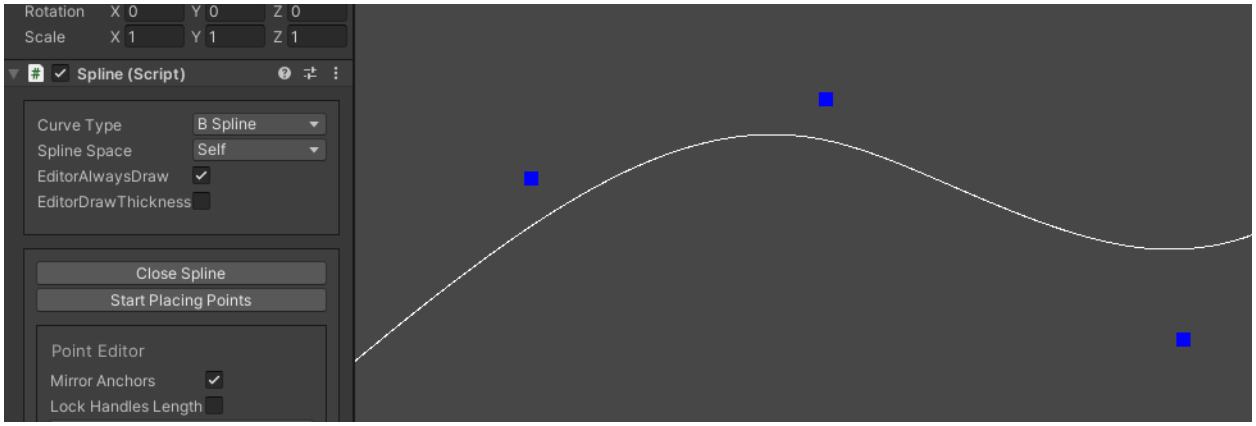
Bezier

The spline is treated as a set of sub-curves, which are each a set of 4 points (2 points, 2 handles). This mode gives more control than BSpline, but is a bit harder to use.



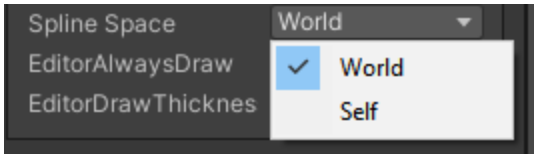
BSpline

The spline is treated as one continuous smooth curve between points.



Spline Space

Splines can be treated as World Space or Local (Self) space. If a Spline is set to World Space, moving around its GameObject will not move the points of the Spline. If it is set to Local (Self) Space, points WILL move and rotate with the GameObject. Try it out!



EditorAlwaysDraw

When enabled, the debug view for the spline will continue to be drawn, even while the GameObject is not selected.

EditorDrawThickness

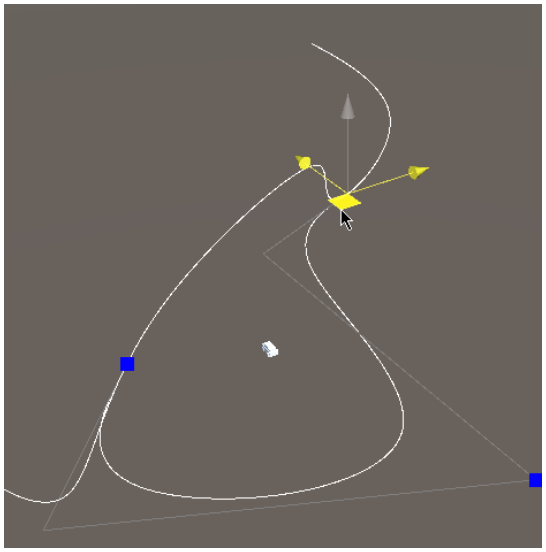
When enabled, the debug view for the Spline will also draw thickness, which is assumed to be the X axis scale between points. This is purely for debug, you can use the scale in any way you feel is useful for your project.

This is also a useful view for viewing the rotation of the spline.

Spline Junctions

Junctions are a tool in the spline system to allow easy blending between splines. If spline A is junctioned to another spline B, any changes to B will be reflected in A, as if it were attached.

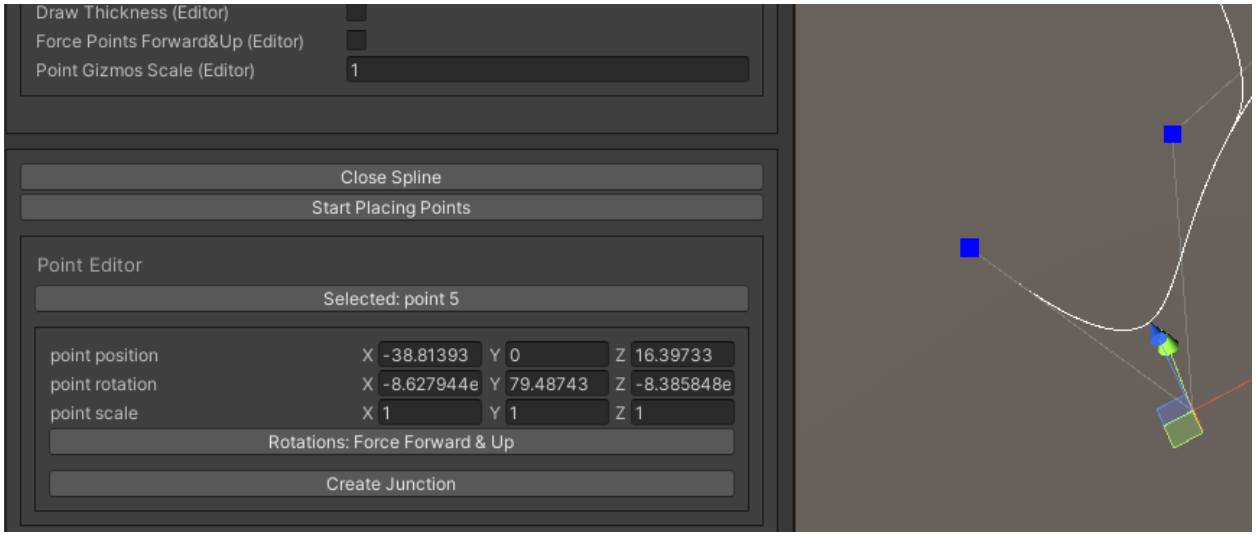
These changes are automatically updated in the Unity editor. At runtime, you will need to call `UpdateJunctions()` on all splines with junctions yourself.



There are two ways to create junctions:

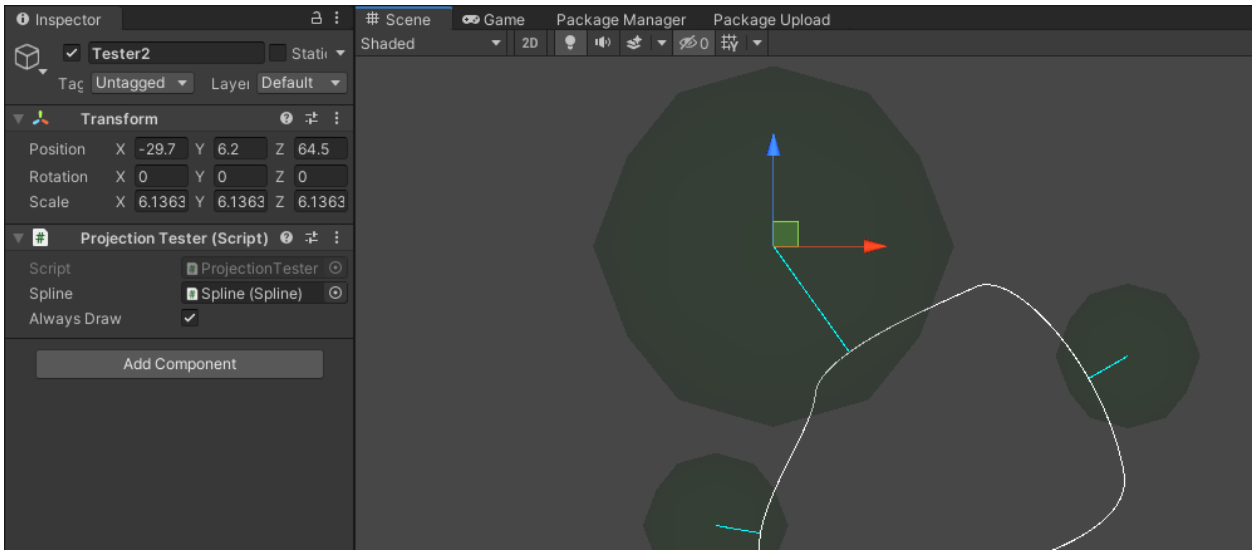
1. Select any point in a spline. In the inspector, press the “Create Junction” button that appears.
2. In any existing spline, you can expand “Junction Tools” at the bottom and drag a new spline reference in. The dragged-in reference will become a dependency for this spline’s start or end junctions.

Please note that the editor will only update junctions correctly for splines that are within the same transform hierarchy (all under the same root are fair game). However, if you are calling UpdateJunctions() yourself, this restriction does not apply.



Simple/Example_ProjectionTester.scene

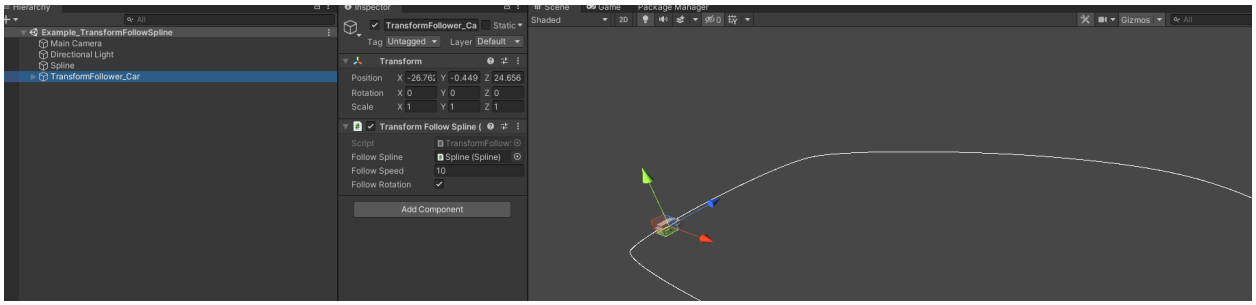
This scene includes the example helper script ProjectionTester, which shows a gizmo view as an example of how to use the Projection API in the Spline class.



You can move around the projection testers and get a feel for how it works.

Simple/Example_TransformFollowSpline.scene

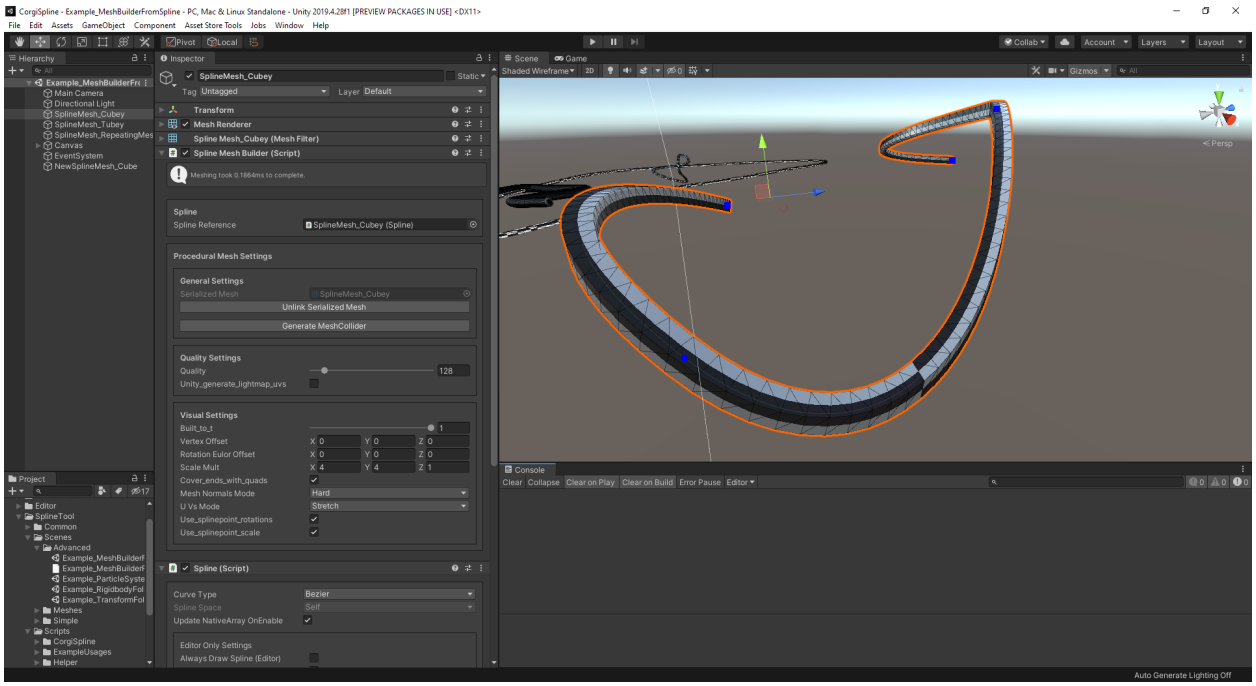
This scene shows a very simple example of a script which can make a single object follow a specified Spline with some simple settings.

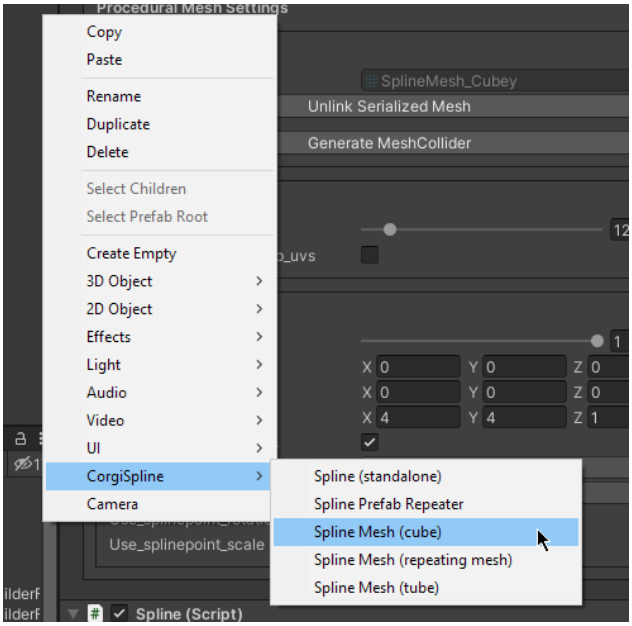


It is only recommended to use this script if you only have a few things to move. If you have a LOT of objects that you want to follow a Spline, I recommend learning about the Unity Job System, and using the Jobified example I've provided. It's located at Advanced/Example_TransformFollowSplineJobified.scene

Advanced/Example_MeshBuilderFromSpline.scene

This scene has three mesh builder scripts. Cubes, Tubes, and Repeatable Meshes.



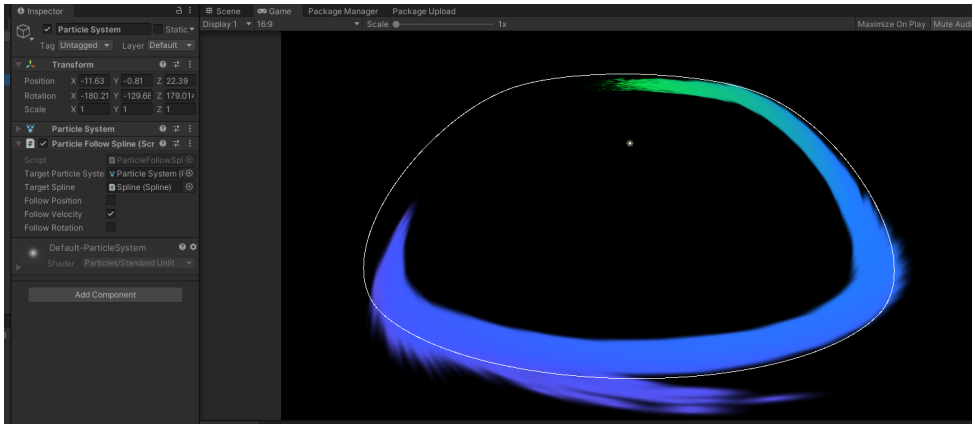


You can create any of these in any scene via the right click menu in the scene hierarchy.

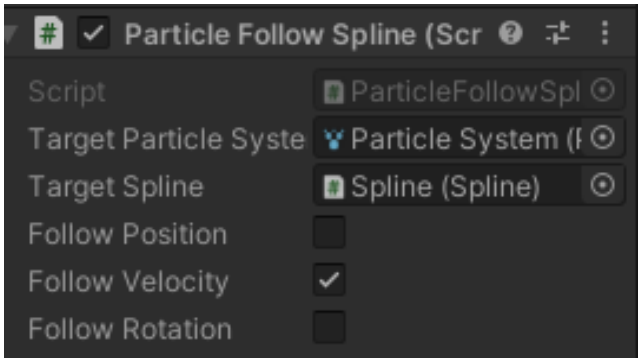
Doing so will automatically create a GameObject with the components setup to let you start creating spline-based meshes right away.

Advanced/Example_ParticleSystemFollowSpline.scene

This scene shows a production-ready example script for having Particles in a ParticleSystem automatically follow a Spline.



To use the script ParticleFollowSpline.cs in your project, you'll have to manually assign the TargetParticleSystem and TargetSpline by dragging in the Unity Inspector, or by assigning it via code.



FollowPosition

This will project the position of every particle directly onto the Spline.

FollowVelocity

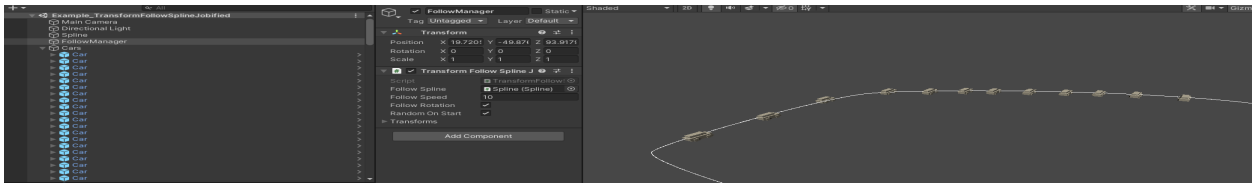
This will project the velocity of every particle onto the forward direction of the Spline, given the particle's current position.

FollowRotation

This will force the rotation of every particle to follow the rotation of the Spline, given the particle's current position.

Advanced/Example_TransformFollowSplineJobified.scene

This scene shows how you can use CorgiSpline to efficiently have a lot of transforms follow a Spline with the same settings. The scene has a TransformFollowManager, which has a list of Transforms attached.



You'll notice that there is a Cars Transform, please read the note attached to it:

```
Note!

FollowMangager's script TransformFollowSplineJobified is internally using
Unity's IJobParallelForTransform job type.

Because of this, for multi-threading to work with transforms, the transforms
must NOT share the same parent.

I'm sharing them here for the sake of the example, with Cars being the parent
transform. At runtime, a helper script attached here will unparent them. This
makes startup time a bit slower for complex scenes, so it is not recommended
to do it this way.

Please remember this!
```

The TransformFollowSplineJobified.cs script takes in this list of Transforms, and first creates a TransformInitializeRandomScatter (implements IJob) to randomly scatter the Transforms, then in Update() it schedules a TransformFollowSplineJob (implements IJobParallelForTransform) to move the Transforms, to follow the Spline.

Please note the following bit of code in the script's Update loop. I'm not .Complete()ing the loop here, which means edits to the native copy of the Spline are not allowed while this is running. If you want to allow editing it while things are following it, simply Complete() here, or, run your edits before this script via Unity's Script Execution Order window.

```
var handle = job.Schedule(_TransformsAccess);
//handle.Complete();
```