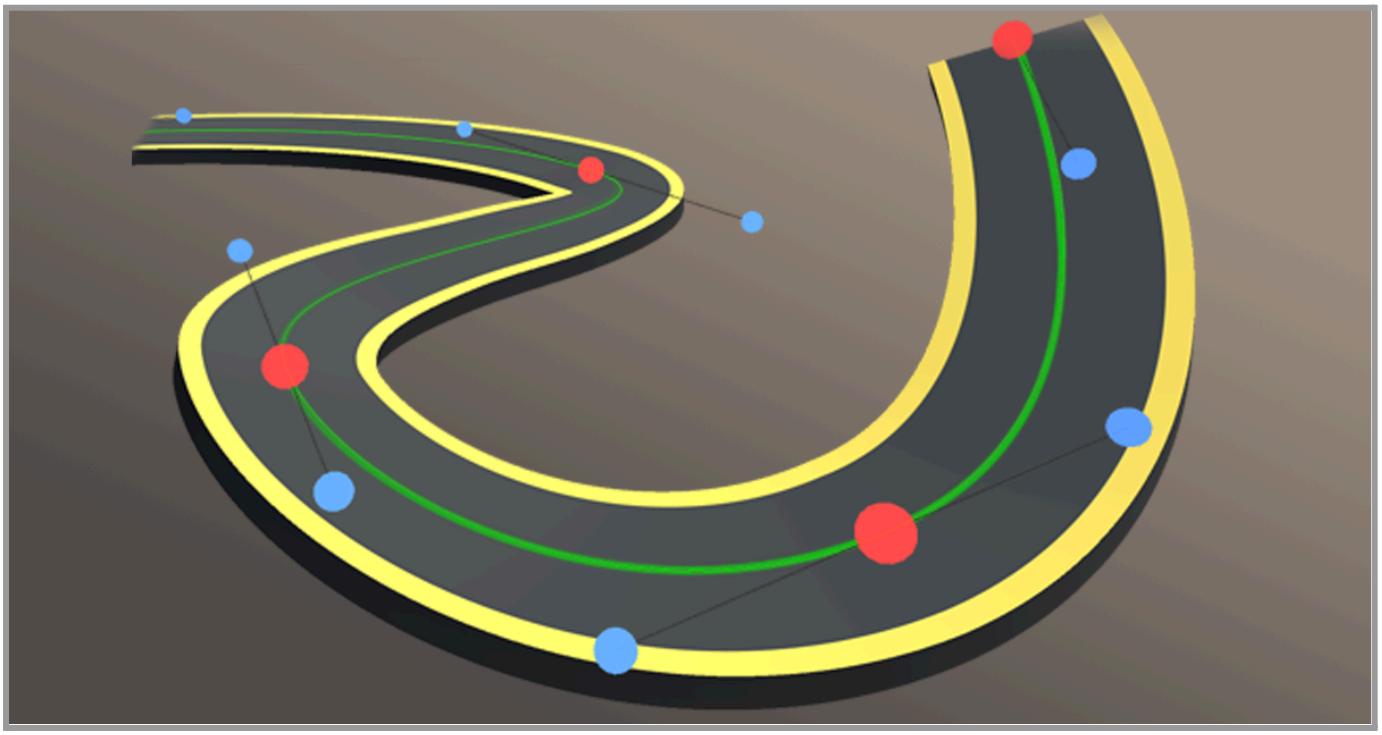
PATH CREATOR

USER GUIDE



Latest version available here: https://github.com/SebLague/Path-Creator

---

Creating a Bézier Path
Scene Controls
Bézier Path Options
Normals Options
Display Options

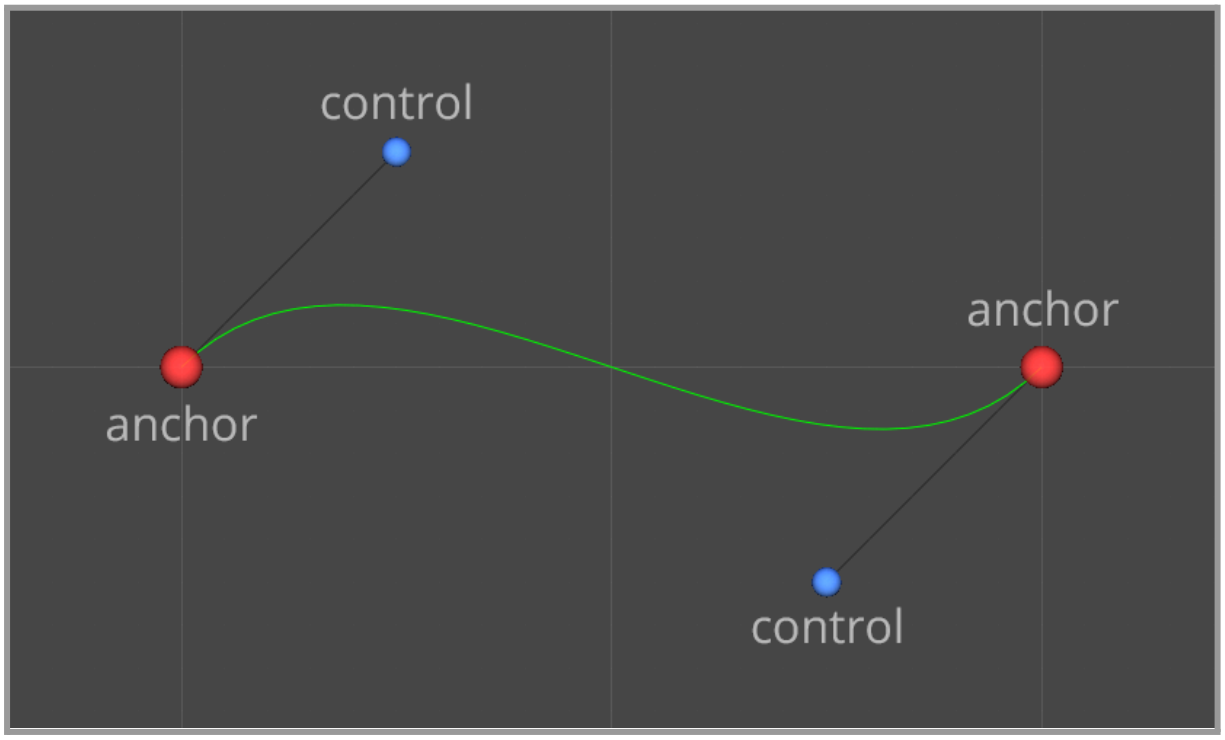
Converting to Vertex Path

Using Paths in a Script
Moving along Path
Generating a Path

Included Examples

Creating a Bézier Path

To begin creating a path, create a GameObject and add the PathCreator script to it. You will see a bézier path appear in the scene.



A bézier path is defined by two types of points: anchors and controls. Anchors are the points which the path actually passes through, while the control points allow you to define the curvature of the path.

Scene Controls:

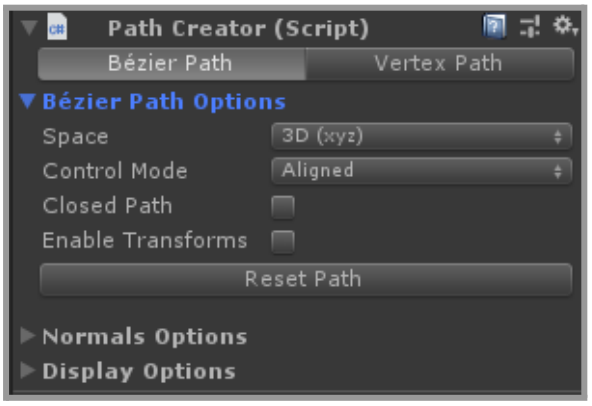
Moving points:
Left-click and drag to move the points around. If you click on a point without dragging, it will turn into a move tool with arrows so that you can move along a single axis.

Adding and Inserting points:
Shift-left-click to add new anchor points to the end of the path (hold the *ctrl* key to add to the start of the path instead). To insert a point, shift-left-click over an existing segment of the path.

Deleting points:
Ctrl-click over an anchor point to delete it.

Editing Normals: (only available for 3D paths)
See the Local Angles section.

Bézier Path Options:

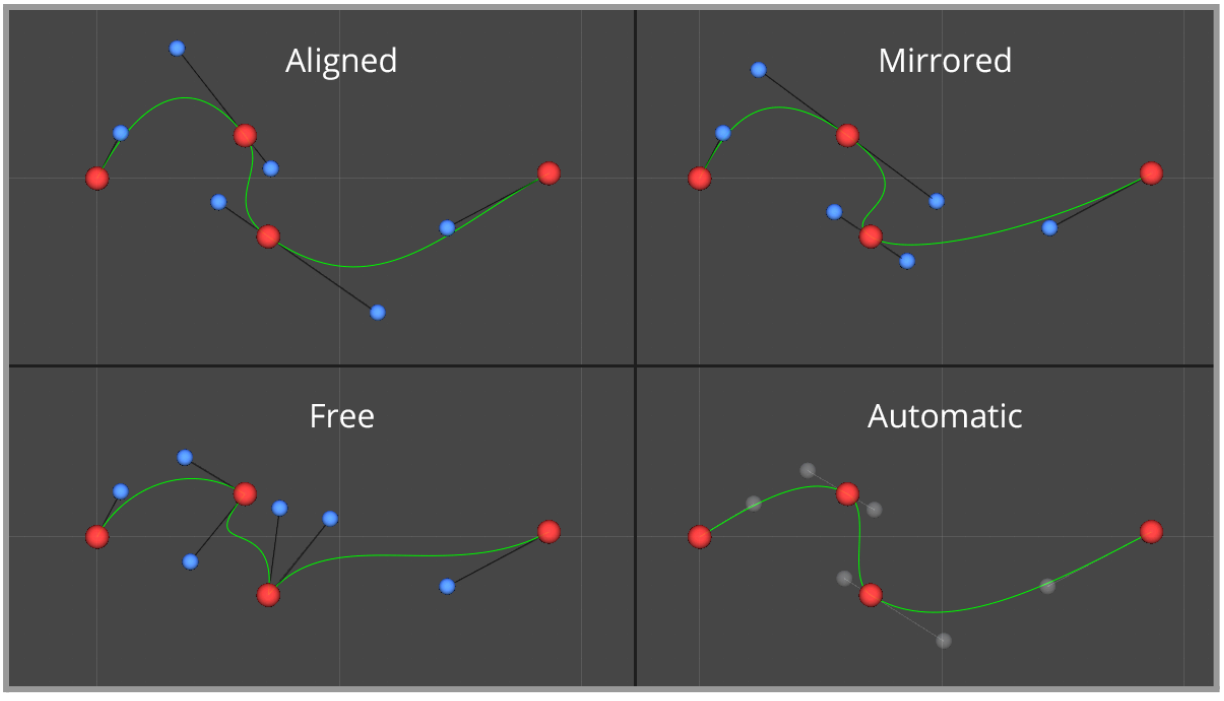


There are two tabs in the PathCreator inspector: **Bézier Path** and **Vertex Path**. The Bézier Path tab is currently selected.

Space:
3D (xyz): points can be positioned freely in 3D space.
2D (xy): points are locked to the xy plane.
Top-Down (xz): points are locked to the xz plane.

Control Mode:
With the exception of the start and end anchors, each anchor has two control points attached to it. How these pairs behave is determined by the control mode.

Aligned: controls stay in a straight line around their anchor point.
Mirrored: controls stay in a straight, equidistant line around their anchor point.
Free: no constraints.
Automatic: controls are positioned automatically to try make the path smooth.

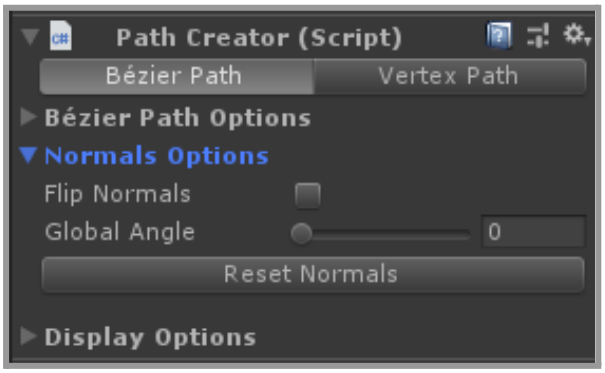


Closed Path:
If set to true, the path will loop around from the end anchor to the start anchor.

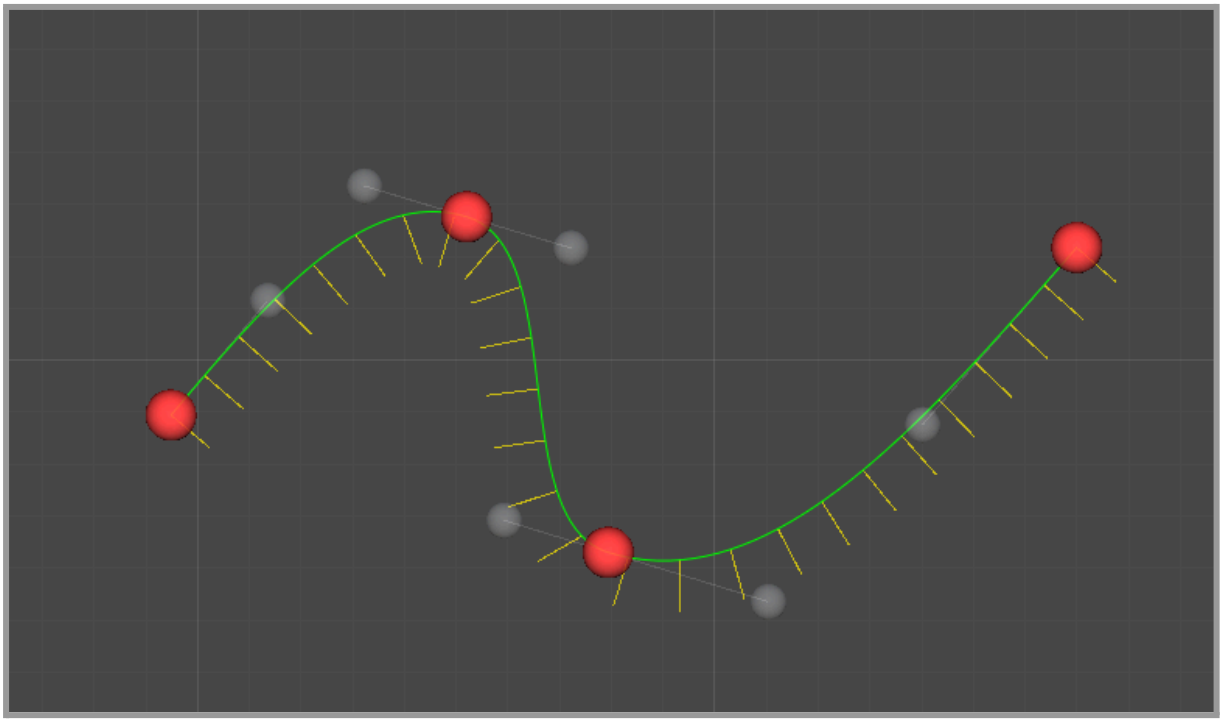
Enable Transforms:
Allows you to move/rotate/scale the path with the standard Unity transform tool.

Reset Path:
This button will reset the path to how it looked at the start.

Normals Options:



Opening the **Normals Options** section will cause the path normals to be displayed in the editor. The normals are vectors perpendicular to the path at each point.

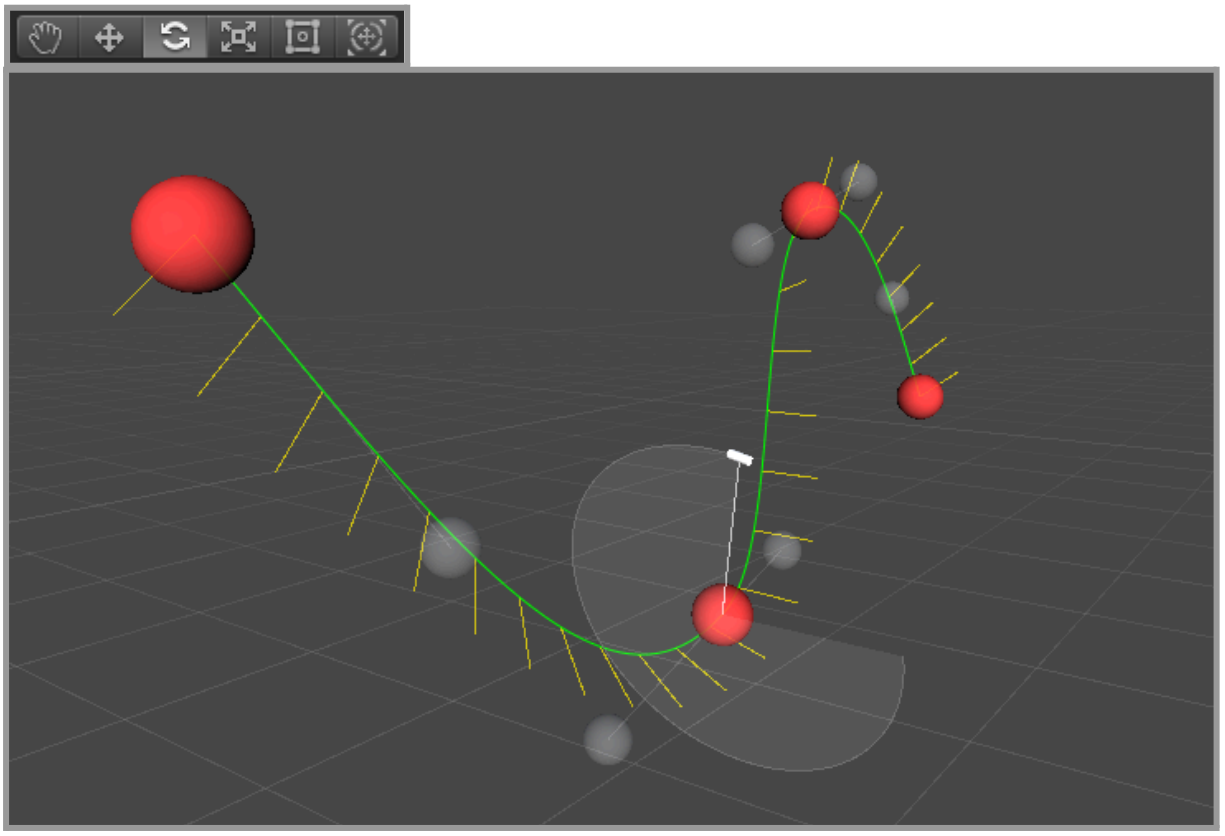


Flip Normals:
This causes the normals to point in the opposite direction.

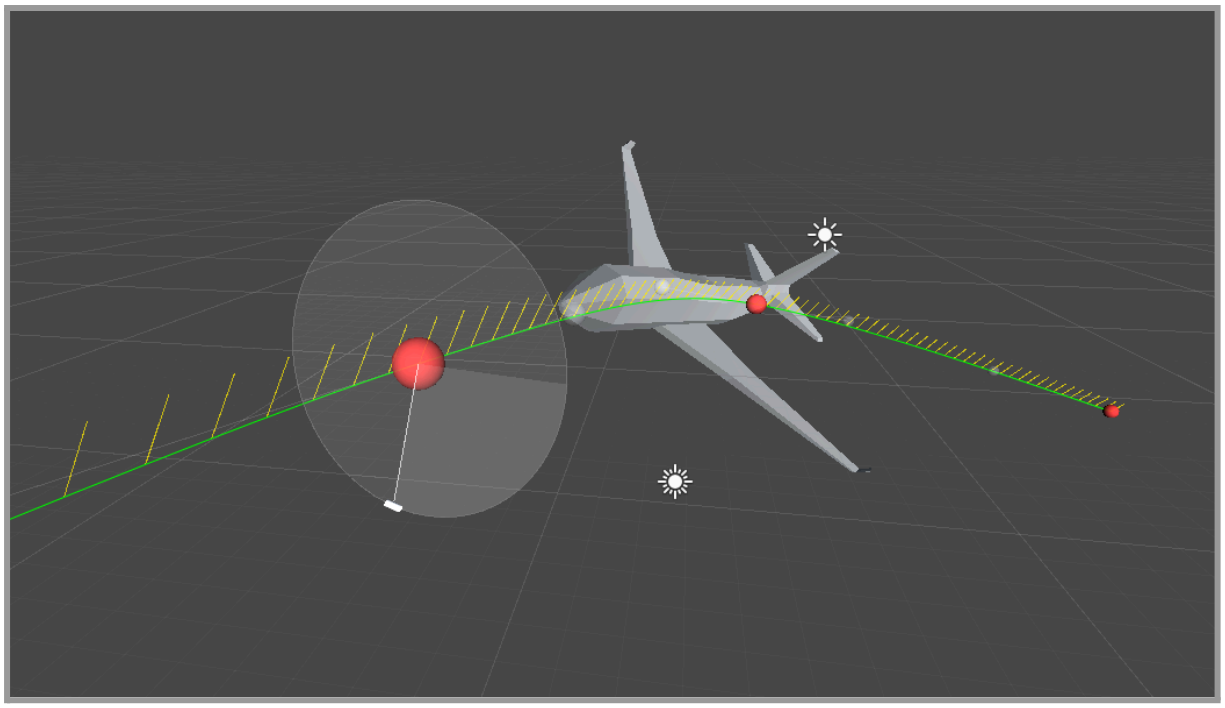
Global Angle: (only available for 3D paths)
Rotates all the normal vectors around the path.

Reset Normals: (only available for 3D paths)
Resets all normals options (Flip Normals, Global Angle, and Local Angles).

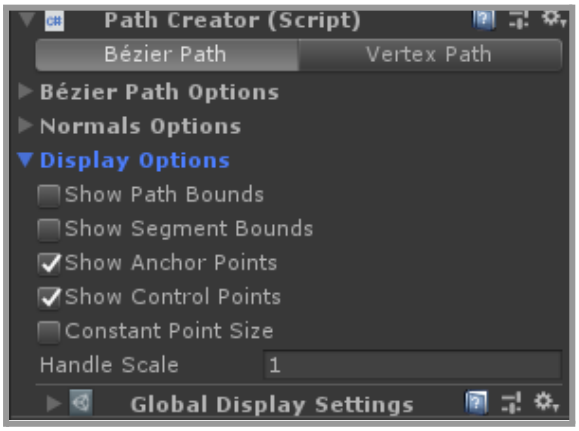
Local Angles: (only available for 3D paths)
With the Normals Options section open in the inspector, a new tool is available in the scene. Select the rotate tool from the standard tools menu in the top left of the editor, and then left click on any anchor point.



This allows you to define the angle of the normals at each anchor point.
This can be useful in a number of cases. For example, imagine you have an aeroplane and you want it to bank as it turns. The normals can define the 'up' direction of the plane as it moves along the path.

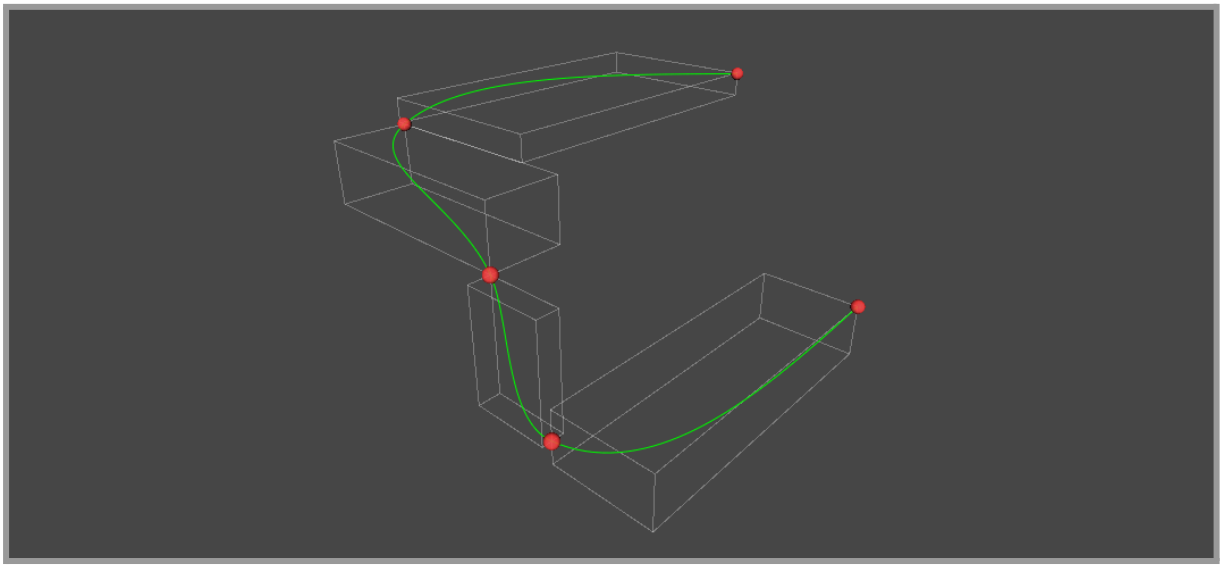


Display Options:



Show Path Bounds:
Draws a bounding box around the path.

Show Segment Bounds:
Draws a bounding box around every segment of the path. This can be helpful for judging depth when working in 3D space.



Show Anchors/Controls:
Toggles the visibility of anchor and control points.

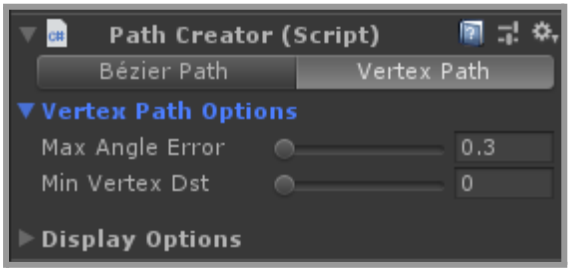
Handle Scale:
Increase/decrease the size of the anchor and control points display.

Constant Point Size:
If true, points will remain the same size when zooming in/out in the scene.

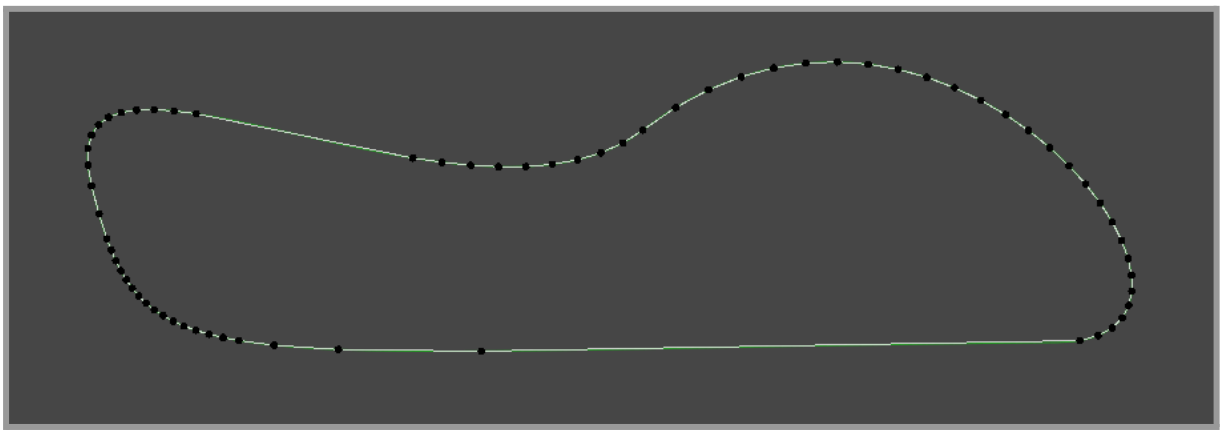
Global Display Settings:
These settings are global, meaning they are used for all path creators.
Here you can customize the colours of the path editor, and change the sizes and shapes of the handles. You can also set whether paths should be visible even when not selected, and if they should be visible when behind objects.

Converting to Vertex Path

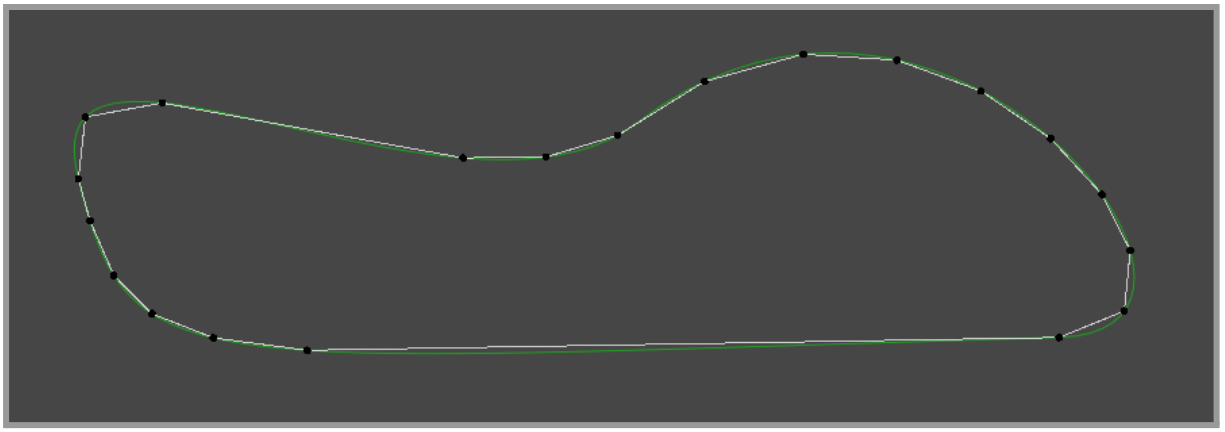
In order to do things like move at a constant speed along a path, the bézier path needs to be converted to a vertex path. This is done automatically, but if you want to customize the settings, you can go into the **Vertex Path** panel of the PathCreator inspector.



In the scene you'll then be able to see where vertices are being placed along the path. Fewer vertices are placed in straight sections of the path to reduce the number of vertices, while still maintaining a high accuracy.



Increasing the **Max Angle Error** or the **Min Vertex Dst** will further reduce the number of vertices, at the cost of accuracy. In general, however, just leaving these at their default values will be perfectly fine.



Using Paths in a Script

Once you've created a path in the editor, you'll probably want to use it in one of your scripts. To do this, you'll first of all need to add the PathCreation namespace. Next, create a public PathCreator variable (called something like pathCreator). You can then access the vertex path by writing pathCreator.path, like in the example below.

```
using UnityEngine;
using PathCreation; // You'll need to include this namespace

public class MyScript : MonoBehaviour
{
   // This needs to be assigned to in the inspector
   public PathCreator pathCreator;

   void Start()
   {
       // You can now access the vertex path with pathCreator.path
       // For example, this sets the position to the middle of the path:
       transform.position = pathCreator.path.GetPoint(0.5f);
   }
}
```

Note that you must assign the PathCreator object that you have in the scene (or a prefab of it) to the pathCreator variable in the inspector.

The VertexPath class has several methods for getting information about the path at a certain 'time' (where **t** = 0 is the start of the path, and **t** = 1 is the end of the path).

GetPoint(float **t**): returns a Vector3 for the position of the path at **t**.
GetDirection(float **t**): returns a Vector3 that points along the path at **t**.
GetNormal(float **t**): returns a Vector3 that points perpendicular to the path at **t**.
GetRotation(float **t**): returns a Quaternion for the rotation of the path at **t**.

All these methods also have an equivalent that take a distance value, instead of a time value. So for example, **GetPointAtDistance**(**30**) will return a Vector3 for the position of the path after 30 units.

All these methods also have an optional second parameter, which is an EndOfPathInstruction. This tells the method what to do if the given time exceeds 1, or the given distance exceeds the length of the path.
Loop: begin again from the start of the path.
Reverse: move backwards from the end to the beginning of the path.
Stop: stop moving at the end of the path.

Moving along a path:

The following example shows how an object can be moved to follow a path at a constant speed. Depending on the 'end' variable, the object will either Loop, Reverse, or Stop upon coming to the end of the path.

```
using UnityEngine;
using PathCreation;

public class PathFollower : MonoBehaviour
{
    public PathCreator pathCreator;
    public EndOfPathInstruction end;
    public float speed;
    float dstTravelled;

    void Update()
    {
        dstTravelled += speed * Time.deltaTime;
        transform.position = pathCreator.path.GetPointAtDistance(dstTravelled, end);
        transform.rotation = pathCreator.path.GetRotationAtDistance(dstTravelled, end);
    }
}
```

Generating a path:

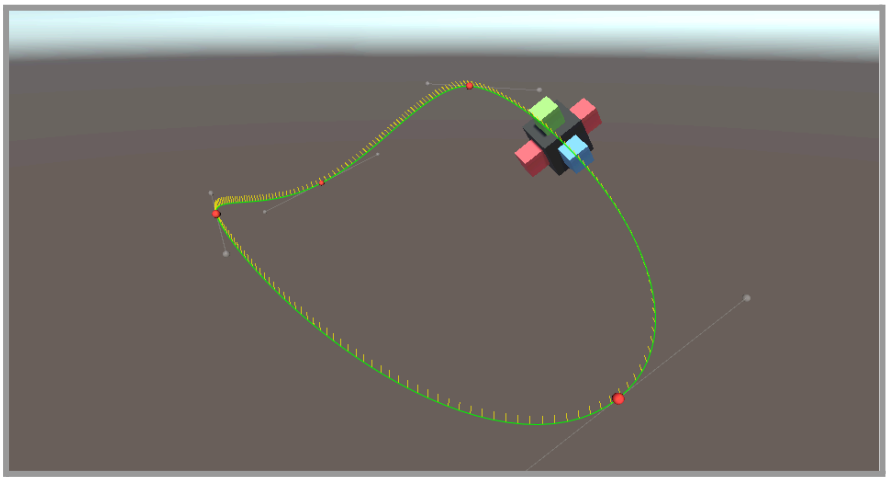
Instead of creating a path in the editor, it is also possible to create a path from script. All you need is an array or list of points, which can be stored as Vector3s, Vector2s, or Transforms. You can pass these into the BezierPath constructor, and it will generate the controls for a smooth path passing through the supplied points. You can then simply create a VertexPath from the BezierPath.

```
VertexPath GeneratePath(Vector2[] points, bool closedPath)
    {
        // Create a closed, 2D bezier path from the supplied points array
        // These points are treated as anchors, which the path will pass through
        // The control points for the path will be generated automatically
        BezierPath bezierPath = new BezierPath(points, closedPath, PathSpace.xy);
        // Then create a vertex path from the bezier path, to be used for movement etc
        return new VertexPath(bezierPath);
    }
```

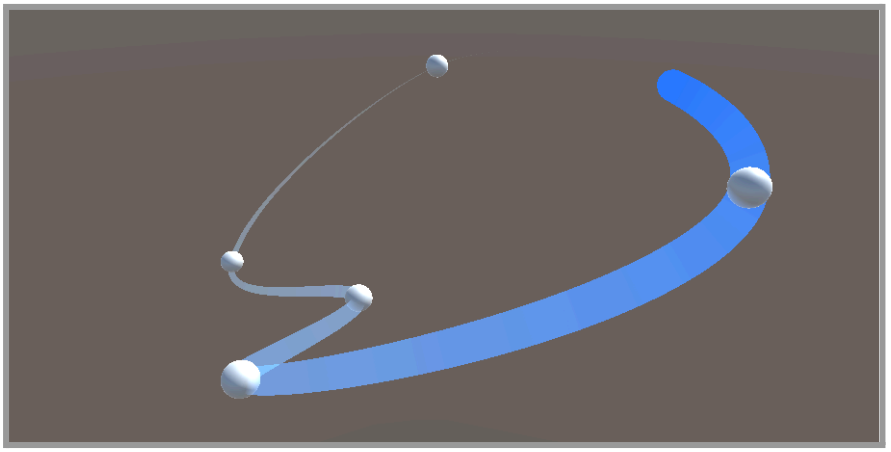
Included Examples

An Examples folder is included in the package, with 3 example scenes. This can safely be deleted if you don't want it.

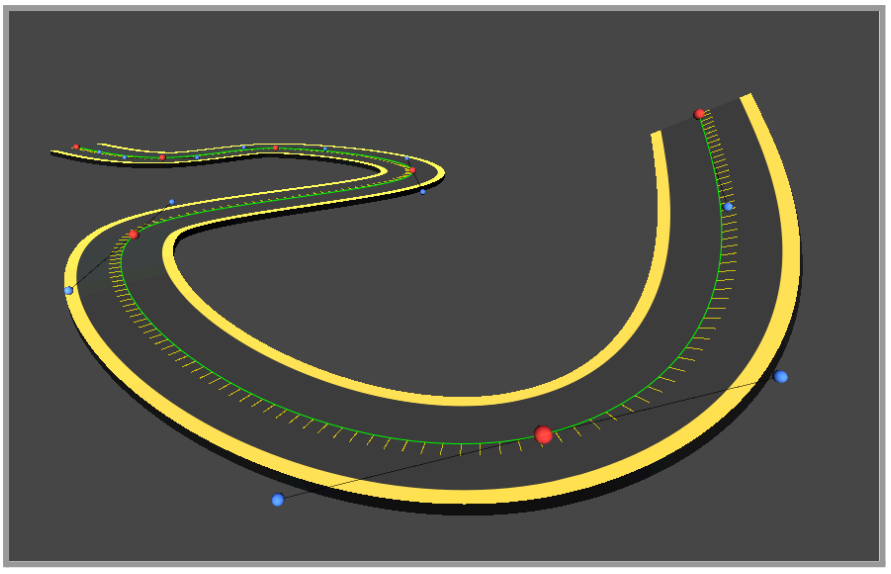
Path Follow shows a simple set-up for having an object move and rotate along a path at a constant speed.



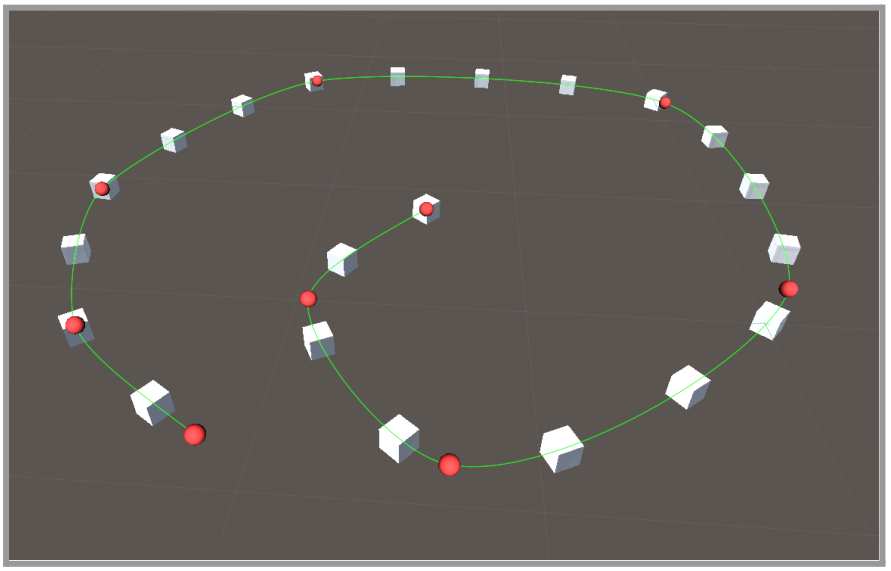
Object Path shows how to create a path from a collection of objects placed in the scene. A GameObject with a trail renderer attached then moves along that path.



Road shows how a procedural mesh can be constructed along the path.



Object Placement shows how to spawn objects along a path at regular intervals.



Path as Prefab shows how to spawn paths from a prefab at runtime.