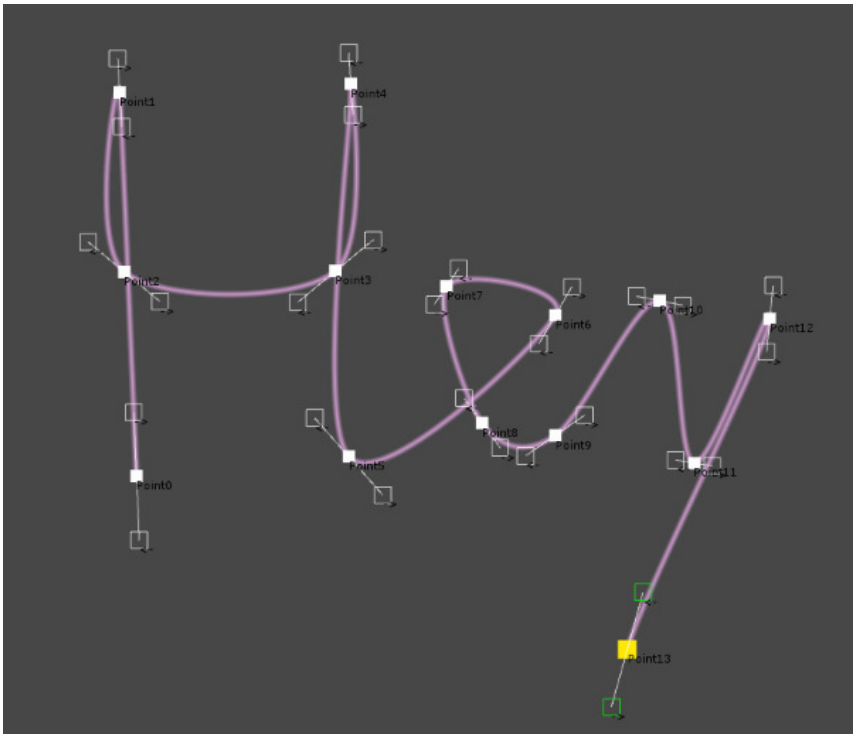
Bezier Spline Solution *by Suleyman Yasir Kula*



Thank you for downloading this asset. This asset is a means to create bezier splines in editor and/or during runtime. Splines can be created and edited visually in the editor, or by code during runtime.

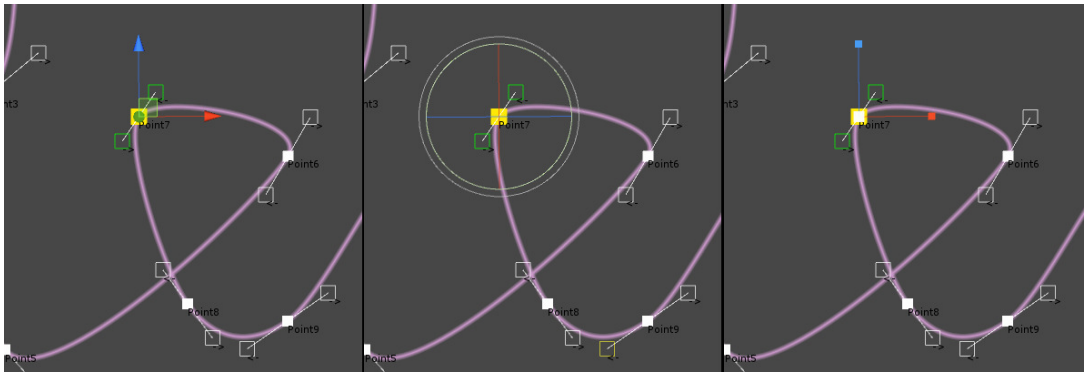
Here, I will briefly talk about the usage of the asset and the important functions that come with it. Enjoy!

Contact: mailto:yasirkula@yahoo.com **Web:** http://yasirkula.net

▪ Creating & Editing a New Spline In Editor

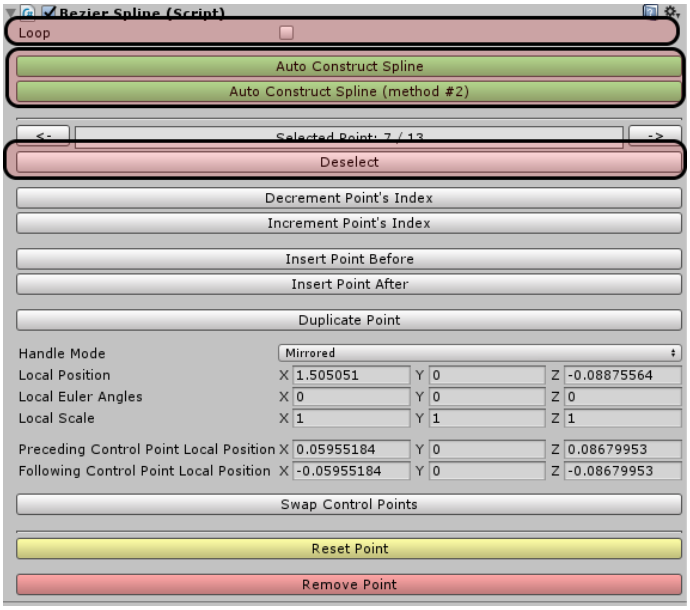
To create a new spline in the editor: "*GameObject - Bezier Spline*"

Now you can select the end points of the spline in the Scene view and translate/rotate/scale or delete/duplicate them as you wish (each end point has 2 control points, which can also be picked and translated):



Food For Thought: Each spline in your scene is in essence a single GameObject with BezierSpline component attached. The position/rotation/scale values for end points are calculated via transformation matrices, rather than using a Transform for each end point. These values are stored in local variables and are recalculated only when necessary.

The user interface for the spline editor should be pretty self-explanatory. However, I'd like to mention a couple of things:



Loop: connects the first end point and the last end point of the spline

Auto Construct Spline: auto adjusts the control points of end points to form a smooth spline that goes through the end points you set. There are 2 different implementations for it, with each giving a slightly different output (see *Fig.1* below)

Deselect: deselects the selected end point. Useful when you want to translate/rotate/scale the spline itself in the Scene view

▪ Creating & Editing a New Spline By Code

- **Create a new bezier spline**

Simply create a new GameObject and attach a BezierSpline component to it:

```
BezierSpline spline = new GameObject().AddComponent<BezierSpline>();
```

- **Create a new end point**

End points must be initialized with a spline reference. There are 3 different ways to create a new end point:

```
// Create a new end point at local position: 0,0,0
BezierPoint point = new BezierPoint( spline );

// Create a new end point at local position: 4,8,15
BezierPoint point2 = new BezierPoint( spline, new Vector3( 4, 8, 15 ) );

// Create a clone (duplicate) of "point" (same position, rotation and scale)
BezierPoint pointClone = new BezierPoint( spline, point );
```

- **Populate the spline**

When the spline is first created, it is automatically initialized with 2 end points (see *Reset()* function of the *BezierSpline*). However, you can initialize the spline with your own end points using Initialize function, if you want. Use InsertNewPointAt to add new end points to the spline and RemovePointAt to remove end points from the spline. There is also a simple swap function to swap the positions of 2 end points:

```
// Initialize the spline with the end points
spline.Initialize( new BezierPoint[] { point, point2, pointClone } );

// Insert a new end point in-between point and point2
spline.InsertNewPointAt( 1 );

// Insert a new end point (with local position 16,23,42) to the end of the spline
spline.InsertNewPointAt( spline.Count, new Vector3( 16, 23, 42 ) );

// Remove the end point at the beginning of the spline
spline.RemovePointAt( 0 );

// Swap the first and the last end points of the spline
spline.SwapPointsAt( 0, spline.Count - 1 );
```

- **Shape the spline**

You can change the position, rotation and scale values of end points and their control points to reshape the spline.

End points have the following properties to store their transformational data: position, localPosition, rotation, localRotation, eulerAngles, localEulerAngles and localScale.

Positions of control points can be tweaked using the following properties in BezierPoint: precedingControlPointPosition, precedingControlPointLocalPosition, followingControlPointPosition and followingControlPointLocalPosition. The local positions are relative to their corresponding end points.

```
// Set first end point's (world) position to 2,3,5
spline[0].position = new Vector3( 2, 3, 5 );

// Set second end point's local position to 7,11,13
spline[1].localPosition = new Vector3( 7, 11, 13 );

// Reposition the control points of the first end point
spline[0].precedingControlPointLocalPosition = new Vector3( 0, 0, 1 );
spline[0].followingControlPointPosition = spline[1].position;
```

- **Auto construct the spline**

If you don't want to position all the control points manually, but rather generate a nice-looking "continuous" spline that goes through the end points you have created, you can call either AutoConstructSpline() or AutoConstructSpline2(). These methods are implementations of some algorithms found on the internet (and credited in the source code). There is a third algorithm (*AutoConstructSpline3()*) which is not implemented, but feel free to implement it yourself!

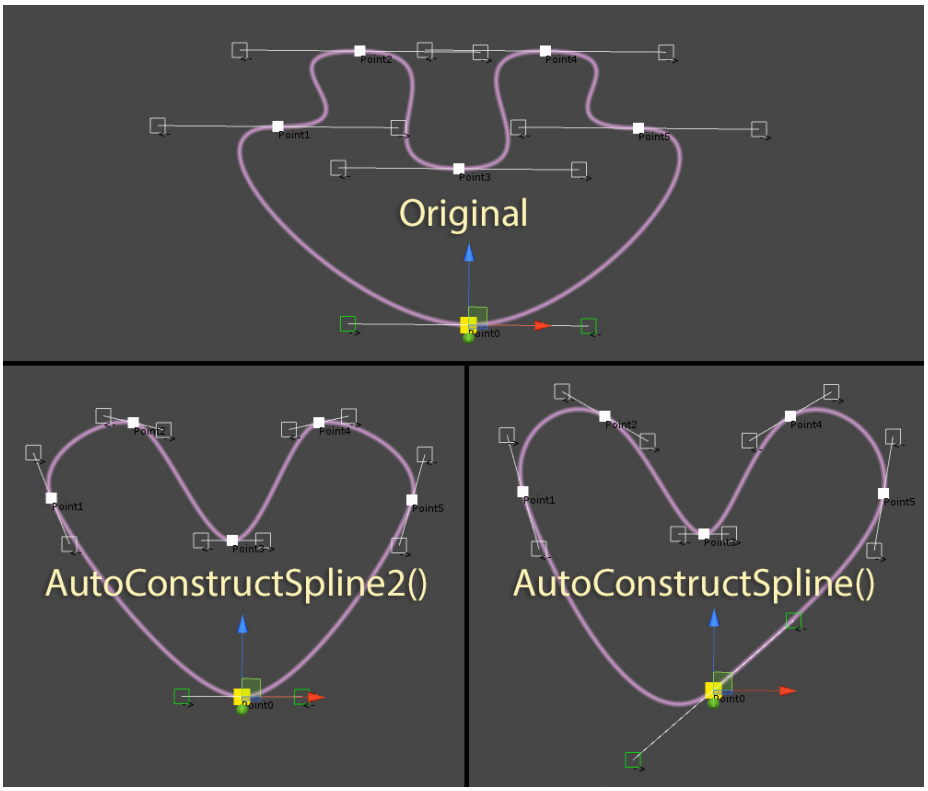


Figure 1: Demonstration of AutoConstructSpline functions

▪ Utility Functions

The framework comes with some utility functions that are not perfect but most of the time, gets the job done. Though, if you want, you can use this framework to just create splines and then apply your own logic to them.

➢ `public Vector3 GetPoint( float normalizedT )`

A spline is essentially a mathematical formula with a [0,1] clamped input (usually called *t*), which generates a point on the spline. As the name suggests, this function returns a point on the spline. As *t* goes from 0 to 1, the point moves from the first end point to the last end point (or goes back to first end point, if spline is looping).

➢ `public Vector3 GetTangent( float normalizedT )`

Tangent is calculated using the first derivative of the spline formula and gives the direction of the movement at a given point on the spline. Can be used to determine which direction an object on the spline should look at a given point.

➢
```
public float GetLengthApproximately( float startNormalizedT, float endNormalizedT,
                                     float accuracy = 50f )
```

Calculates the approximate length of a segment of the spline. To calculate the length, the spline is divided into "accuracy" points and the Euclidean distances between these points are summed up.

Food For Thought: BezierSpline has a Length property, which is simply a shorthand for "GetLengthApproximately(0f, 1f)".

➢
```
public Vector3 FindNearestPointTo( Vector3 worldPos, out float normalizedT,
                                   float accuracy = 100f )
```

Finds the nearest point on the spline to any given point in 3D space. The normalizedT parameter is optional and it returns the parameter *t* corresponding to the resulting point. To find the nearest point, the spline is divided into "accuracy" points and the nearest point is selected. Thus, the result will not be 100% accurate most of the time but good enough for most of the casual use-cases.

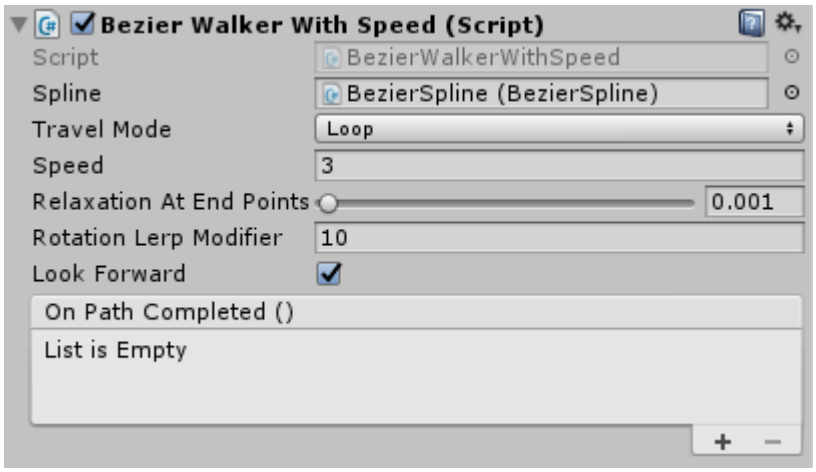
➢
```
public Vector3 MoveAlongSpline( ref float normalizedT, float deltaMovement,
                                bool increasedAccuracy = false,
                                int maximumNumberOfChecks = 20,
                                float maximumError = 0.001f )
```

Moves a point (normalizedT) on the spline deltaMovement units ahead and returns the resulting point. The normalizedT parameter is passed by reference to keep track of the new *t* parameter. To calculate the resulting point, a binary search-like algorithm is used. Maximum number of trials is defined with the maximumNumberOfChecks parameter. When the distance between the original point and a candidate point is smaller than maximumError, or maximumNumberOfChecks is reached, function stops and returns the closest point found. If increasedAccuracy is false, distance between the original point and a candidate point is calculated using Euclidean distance. Otherwise, the distance is calculated using the GetLengthApproximately function.

▪ Other Components

Framework comes with 2 additional components that may help you move objects along the spline. These components are located in the Utilities folder.

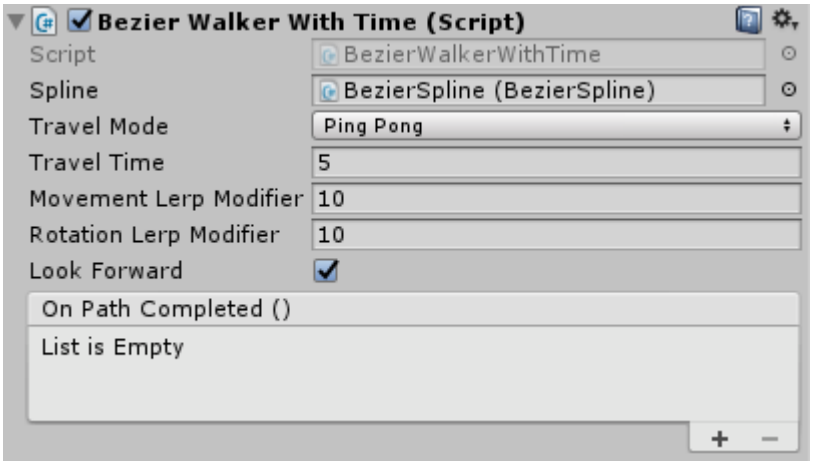
- **BezierWalkerWithSpeed**



Moves an object along the spline with constant speed. There are 3 travel modes: Once, Ping Pong and Loop. If Look Forward is selected, the object will always face forward and the smoothness of the rotation can be adjusted using the Rotation Lerp Modifier. Each time the object completes a lap, its On Path Completed () event is invoked.

If the object is moving at high speed, it sometimes slows down briefly near the last end point. This effect can be counteracted by increasing the Relaxation At End Points. However, if its value is too high, then the object starts skipping a small segment of the spline near the last end point.

- **BezierWalkerWithTime**



Travels the spline in Travel Time seconds. Movement Lerp Modifier parameter defines the smoothness applied to the position of the object.